Local Graph Clustering Using ℓ_1-regularized PageRank Algorithms

by

Chufeng Hu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer science

Waterloo, Ontario, Canada, 2020

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Local graph clustering methods are used to find small- and medium-scale clusters without traversing the graph. It has been shown that the combination of Approximate Personalized PageRank (APPR) algorithm and sweep method can efficiently detect a small cluster around the starting vertex [2]. This research explores the optimization framework proposed in the work by Fountoulakis et al. [15], where a connection between the APPR and an ℓ_1-regularized objective function is revealed. We propose a coordinate descent method for solving the ℓ_1-regularized PageRank problem. We prove that our method has running time dependent on the number of nonzero coordinates in the optimal solution. In addition, we compare 6 optimization algorithms for solving the ℓ_1-regularized PageRank problem in large graphs. We demonstrate that the proposed coordinate descent outperforms the original proximal gradient descent, and the accelerated first order algorithms have the best performance among all algorithms measured in our experiment.

Acknowledgements

I would like to thank all the people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A graph consists of vertices. The connection between two vertices is called edge, which indicates some kind of relationship between the vertices. Figure 1.1 shows a small graph with 8 vertices.

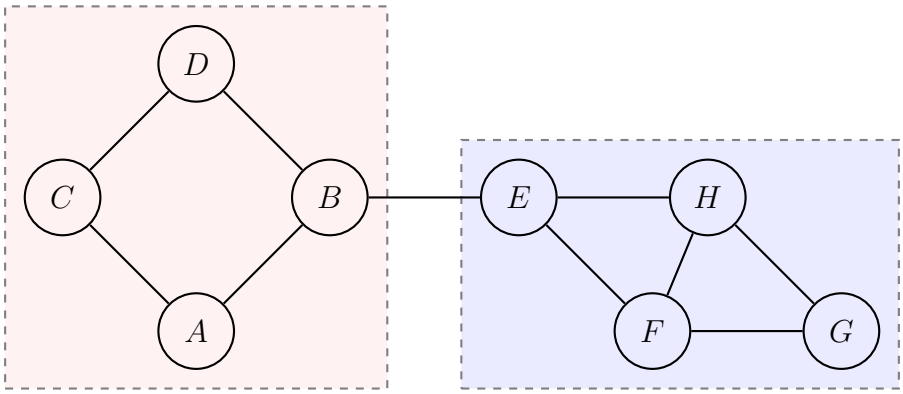

Figure 1.1: A graph with 8 vertices.

Graph representation is simple but powerful. If we want to find the fastest way from Waterloo to Toronto, then we may use Dijkstra's shortest path algorithm. If we want to connect two clusters with minimum cost, we can apply minimum spanning tree algorithm. If we want to recommend friends to users, we can group people in the network and introduce them to each other if they are in the same group.

In modern graph analysis, applications need to handle very large graphs and algorithms that iterating the whole graphs can be very expensive or infeasible [36]. Additionally, many

graphs don't have a clear global structure and data can be noisy. This means vertices can belong to multiple clusters and clusters' boundaries are hard to detect [14].

Local clustering methods are used to to identify a cluster containing the specific seed vertices, which are the ones we care about. Using local clustering algorithms to find a desired cluster usually requires a small subset of vertices to be visited. Therefore, the scalability problem in a large graph can be avoided, as the whole graph does not need to be processed. In addition, clusters with different starting vertices can be simultaneously obtained by parallel computation [37]. These advantages make local clustering methods more applicable for today's large-scale graphs.

Figure 1.2 shows the result of local graph clustering method in the graph of USA road system [6]. The number of visited vertices during the computation is independent of the size of the graph. Figure 1.3 shows the clustering results in a protein-protein interactions graph [25]. Local graph clustering method finds all clusters that have at least 5 vertices.

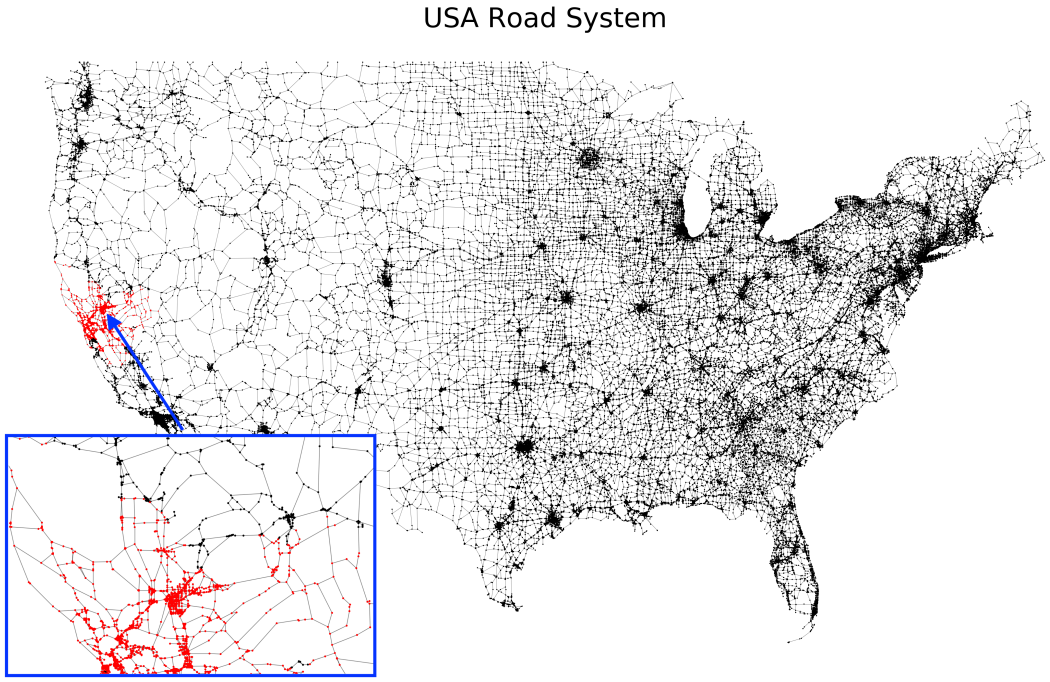

Figure 1.2: This figure shows the clustering result in the graph of USA road system [6]. The local graph clustering method visits a subset of the graph and returns a single cluster (i.e. red points).

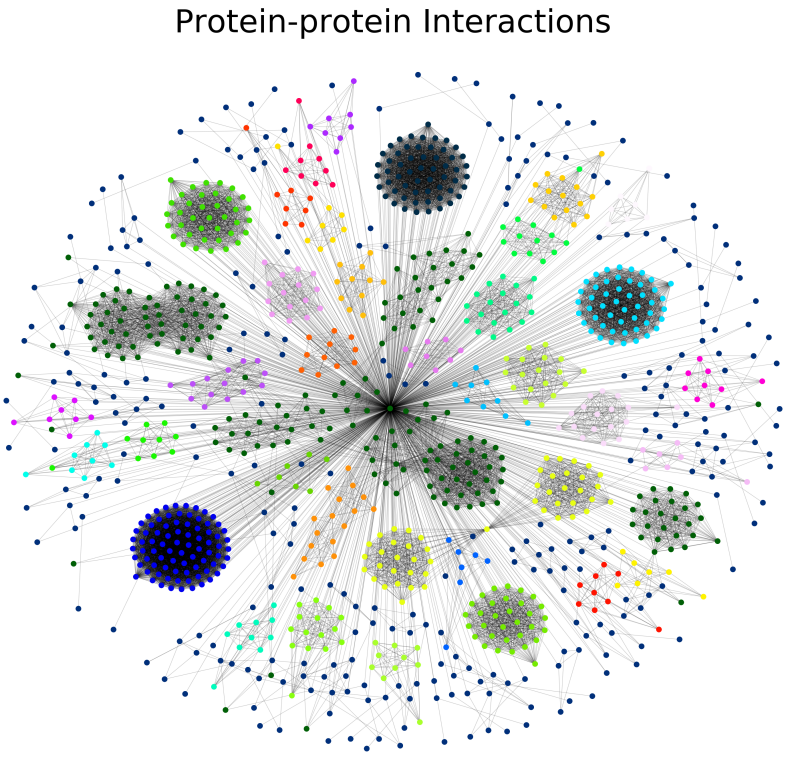

Figure 1.3: Graph clustering method finds 36 clusters in a protein-protein interactions graph [25].

1.1 Related Work

Scalability problems in large-scale graphs lead to the development of local approaches, which focus on a relatively small subset of a graph. Most of local graph clustering methods return a single cluster around a given seed vertex or a set of seed vertices.

Mahoney [24] introduces **LocalSpectral** that adopts ideas from spectral graph theory [38]. **LocalSpectral** is designed to construct a locally-biased analogue of the second

eigenvalue and its associated eigenvector of the Laplacian matrix, which are the two fundamental objects of the graph [10]. Their research is the very first one that is derived from an explicit optimization problem. Interestingly, the analysis shows the optimal solutions to **LocalSpectral** are generalizations of the Personalized PageRank [7, 24].

Different from spectral methods, **LocalImprove** is a flow-based algorithm introduced by Orecchia and Zhu [32]. This method is a local formulation of the **Improve** method [23]. Their local procedure is designed to find a low-conductance cut among the cuts near the seed vertex. More specifically, **LocalImprove** method uses a local procedure to solve the cut-improvement problem. The cut-improvement is a problem framework, where we are given an initial partitioning of a graph and the goal is to identify a better cut with respect to the quotient of the cut and size [43].

Cui et al.[11] have done interesting research on the community search problem, where the goal is to find a community whose vertices are close to each other. The closeness is measured by the community's *minimum degree* [11]. To optimize the *minimum degree* metric, their **local search** strategy does a search in the neighborhood of the seed vertex and eventually returns a single cluster.

MQI is introduced by Lang and Rao [21]. Given a cut of a graph and a *quotient cut score* q [21], **MQI** monotonically improves the score q by looking at any subsets of the original cut. Thus operationally, **MQI** is always a local algorithm since it only considers vertices within the origin set.

Spielman and Teng design a local clustering method called **Nibble** [40], which makes a novel use of random walks. **Nibble** finds a cluster whose internal connections are significantly richer than its external connections near a given vertex. The running time of **Nibble**, when it finds a non-empty local cluster, is nearly linear in the size of the output cluster [40].

APPR is first introduced by Andersen et al. [2] to find an approximation of the PageRank vector, since then it becomes the cornerstone of local clustering algorithms. In Chapter 3, we will review **APPR** and explain that how its result can be used in graph clustering.

1.2 Notation

In this thesis, vectors are denoted by lowercase letters, for example $x \in \mathbb{R}^n$ is an n dimensional vector. Matrices are denoted by uppercase letters, for example $A_{m\times n}$ is an m by n

matrix. The i-th coordinate of a vector x is denoted by x_i. The element at i-th row and j-th column of a matrix A is denoted as A_{ij}. The iteration counter is denoted by k and is placed as superscript with parentheses, for example, $q^{(k)}$ is the vector at iteration k.

The dot product of two vectors is written as $\langle x, y\rangle = x^T y$, where x^T is the transpose of x. The vector of all ones is denoted as e. The vector, whose i-th coordinate is one and zero elsewhere, is denoted as $e(i)$. Define $[n] := \{1, 2, \ldots, n\}$, where the integer $n > 1$. The support set of a vector $x \in \mathbb{R}^n$ is denoted as

$$supp(x) := \{i \in [n] | x_i \neq 0\}.$$

A comparison between a vector x and a scalar α means the comparison between every entry of x and α, for example, $x > 0$ means $x_i > 0\ \forall i$.

A multivariate function f that takes n variables and returns a scalar is denoted as $f : \mathbb{R}^n \to \mathbb{R}$. The domain of the function f is denoted as $\mathbf{dom} f$. For any point $x \in \mathbf{dom} f$, the gradient of function f is written as $\nabla f(x)$, and $\nabla_i f(x)$ is the i-th coordinate of the gradient.

A graph $G = (V, E)$ is a pair of sets, where V is the set of vertices and E is the set of edges. For simplicity, we give each vertex a unique ID from 1 to m, thus a vertex $i \in [m]$.

Throughout this thesis, we assume graphs are undirected, unweighted and without self-loops (i.e. no vertex is the neighbor of itself). An edge can be written as an unordered pair $\{i, j\}$, where the vertices i and j are endpoints. If there exists an edge $\{i, j\} \in E$, we say that i is a neighbor of j and denote it as $i \sim j$. For any nonempty set of vertices C, the relation $i \sim C$ means that vertex i is a neighbor of at least one vertex in C. The number of vertices in C is denoted as $|C|$.

The set of neighbors of the vertex j is called the neighborhood of j and written as $\Gamma(j) := \{i | i \sim j\}$. The degree of the vertex j is denoted as $d_j := |\Gamma(j)|$. The volume of a vertices set C is

$$vol(C) := \sum_{i \sim C} d_i.$$

Let D be the diagonal degree matrix of graph G, then

$$D = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_m \end{bmatrix}.$$

The adjacency matrix A of a graph G is an $m \times m$ matrix where

$$A_{ij} = \begin{cases} 1, & \text{if } i \sim j, \\ 0, & \text{otherwise.} \end{cases}$$

1.3 Contribution

We study the local graph clustering methods that take advantage of Approximate Personalized PageRank (APPR) vector, which can be computed by APPR, Algorithm 4. Fountoulakis et al. [15] propose an optimization framework that draws a connection between APPR and an ℓ_1-regularized function. It has been shown that the proximal gradient descent method solves the ℓ_1-regularized problem in a localized manner as the number of nonzero entries in the iterates is independent of the size of the graph [15].

However, the proximal gradient descent method does not take into account of the fact that the gradient of the objective function $f(q)$ defined in (4.1) is coordinate-wise Lipschitz continuous, i.e. for $q \in \mathbb{R}^m$,

$$|\nabla_i f(q + te(i)) - \nabla_i f(q)| \leq L_i |t|, \quad \forall i \in [m] \text{ and } t \in \mathbb{R},$$

where $L_i > 0$ is the coordinate-wise Lipschitz constant. We present a coordinate descent method to solve the ℓ_1-regularized problem while utilizing the coordinate-wise Lipschitz constant. The main challenge is to prove that the iteration complexity of our method depends on the support of the optimal solution. Our contributions can be summarized as follows:

- We prove that the proposed randomized coordinate descent only updates vertices in a subset of the graph when solving the ℓ_1-regularized PageRank problem defined in (4.3).
- We derive the iteration complexity of the proposed randomized coordinate descent when the objective function is α-strongly convex.
- We apply accelerated first order methods to solve the ℓ_1-regularized PageRank problem.
- We implement 6 optimization algorithms in C++ and compare their performances in large-scale graphs.

Chapter 2

Optimization Background

The iterative optimization algorithms produce a sequence $\{x^{(k)}\}$ that eventually converges to the minimum point of an objective function. The computation in iteration k can be generalized into the following formula

$$x^{(k)} = x^{(k)} - t^{(k)}\Delta x^{(k)},$$

where $\Delta x^{(k)}$ is the update direction and $t^{(k)}$ is the step size. The iterative optimization algorithms have the general procedure as algorithm 1.

Algorithm 1 General iterative optimization algorithm

```
1: procedure MINIMIZE(x^(0))
2:     for k = 0, 1, ..., K − 1 do
3:         Find a direction Δx^(k);
4:         Choose a step size t^(k) > 0;
5:         x^(k+1) ← x^(k) − t^(k) Δx^(k);
6:     return x^(K);
```

2.1 Proximal Gradient Descent

Proximal gradient descent is a common method for solving non-differentiable convex optimization problems. First, let us consider the minimization of a differentiable function $f : \mathbb{R}^m \to \mathbb{R}$

$$\min_x f(x).$$

The gradient descent method with a step size $t^{(k)}$ generates the sequence of $\{x^{(k)}\}$ via

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f(x^{(k)}),$$

which can be rewritten in the following way

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ f(x^{(k)}) + \langle x - x^{(k)}, \nabla f(x^{(k)}) \rangle + \frac{1}{2t^{(k)}} \|x - x^{(k)}\|_2^2 \right\}. \tag{2.1}$$

The right hand side of (2.1) can be interpreted as a minimization of a function that approximates $f(x)$. Let us consider an ℓ_1-regularized problem:

$$\min_x f(x) + \lambda \|x\|_1,$$

where λ is the regularization parameter. Adopting a quadratic approximation, one can update $x^{(k)}$ by

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ f(x^{(k)}) + \langle x - x^{(k)}, \nabla f(x^{(k)}) \rangle + \frac{1}{2t^{(k)}} \|x - x^{(k)}\|_2^2 + \lambda \|x\|_1 \right\}, \tag{2.2}$$

which can be further simplified to

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ \lambda \|x\|_1 + \frac{1}{2t^{(k)}} \|x - (x^{(k)} - t^{(k)} \nabla f(x^{(k)}))\|_2^2 \right\}. \tag{2.3}$$

Now, we consider a more general model that minimizes the function $F : \mathbb{R}^m \to \mathbb{R}$, which is the summation of a differentiable function $f : \mathbb{R}^m \to \mathbb{R}$ and a non-differentiable function $g : \mathbb{R}^m \to \mathbb{R}$.

$$\min_x F(x) = f(x) + g(x) \tag{2.4}$$

For simplicity, we make the following assumptions

- $f(x)$ is convex, smooth and differentiable with Lipschitz constant L.
- $g(x)$ is convex but not necessarily smooth.
- The function F defined in (2.4) is lower bounded.

Similar to (2.3), we solve (2.4) by updating the sequence $\{x^{(k)}\}$ as follows

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ f(x^{(k)}) + \langle x - x^{(k)}, \nabla f(x^{(k)}) \rangle + \frac{L}{2} \|x - x^{(k)}\|_2^2 + g(x) \right\}. \tag{2.5}$$

Again, (2.5) can be simplified to

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ g(x) + \frac{L}{2}\|x - (x^{(k)} - \frac{1}{L}\nabla f(x^{(k)}))\|_2^2 \right\} \tag{2.6}$$

Denote the proximity operator of a convex function h as

$$\mathbf{prox}_{h,t}(x) := \underset{u}{\operatorname{argmin}} \left\{ h(x) + \frac{1}{2t}\|u - x\|_2^2 \right\},$$

where $t > 0$. Then it is easy to see that (2.6) is equivalent to

$$x^{(k+1)} = \mathbf{prox}_{g,\frac{1}{L}} \left(x^{(k)} - \frac{1}{L}\nabla f(x^{(k)}) \right).$$

Finally, we get the proximal gradient method [35] in Algorithm 2 for solving (2.4).

Algorithm 2 Proximal gradient method

```
procedure MINIMIZE(x_0, L)
    for k = 0, 1, ..., K − 1 do
        Compute ∇f(x^(k));
        x^(k+1) ← prox_{g,1/L}(x^(k) − (1/L)∇f(x^(k)));
    return x^(k);
```

2.2 Mirror Descent

In this study, we focus on the mirror descent method that has been modified for solving a non-differentiable function. Mirror descent is proposed by Nemirovski and Yudin [5], the modified mirror descent discussed in this section is first introduced in the work of Duchi et al [12].

In the proximal gradient, we make an approximation with a quadratic function plus a non-differentiable function (e.g. ℓ_1 norm). However, the quadratic term could be improved. Let us rewrite the approximation in (2.5) as

$$x^{(k+1)} = \underset{x}{\operatorname{argmin}} \left\{ f(x^{(k)}) + \langle x - x^{(k)}, \nabla f(x^{(k)}) \rangle + D(x, x^{(k)}) + g(x) \right\},$$

where $D(x, x^{(k)}) = \frac{L}{2}\|x - x^{(k)}\|_2^2$ is the quadratic term. In mirror descent, $D(x, x^{(k)})$ is set to be the Bregman divergence with respect to some function ψ

$$D(x, x^{(k)}) := \psi(x) - \psi(x^{(k)}) - \langle x - x^{(k)}, \nabla\psi(x^{(k)})\rangle.$$

Note that setting $\psi(x) = \|x\|_2^2$ leads us back to the proximal gradient method where $D(x, x^{(k)}) = \|x - x^{(k)}\|_2^2$. It also generalizes to many other useful distances, such as KL-divergence when

$$\psi(x) = \sum_{i=1}^{m}(x_i \log x_i - x_i), \quad x \in \mathbb{R}^m,$$

where x is the vector of a probability distribution. Mirror descent has been shown to work well with the optimization problems, where the coefficient vector represents a probability distribution [1].

2.3 Accelerated Proximal Gradient Descent

Beck and Teboulle [3] propose an accelerated method that evaluates the gradient step at an extrapolated point $y^{(k)}$ rather than $x^{(k)}$. Algorithm 3 presents the accelerated method for solving the problem (2.4). It is also called the fast iterative shrinkage thresholding algorithm (FISTA).

Nesterov's work [27] is one of the first few studies that accelerate the gradient descent method, and FISTA is a follow-up research on this topic. These accelerated schemes achieve optimal convergence rate among all first-order methods [27, 3]. While being simple to implement, it is not easy to have an intuitive understanding of these accelerated methods. Recently a connection to differential equation was shown in [41].

Algorithm 3 Accelerated Proximal Gradient Descent (FISTA)

1: **procedure** MINIMIZE($x^{(0)}, L$)
2: $t^{(0)} \leftarrow 1$;
3: $y^{(0)} \leftarrow x^{(0)}$;
4: **for** $k = 0, \ldots, K-1$ **do**
5: $x^{(k+1)} \leftarrow \mathbf{prox}_{g(x), \frac{1}{L}}(y^{(k)} - \frac{1}{L}\nabla f(y^{(k)}))$;
6: $t^{(k+1)} \leftarrow \frac{1+\sqrt{1+4(t^{(k)})^2}}{2}$;
7: $\beta^{(k)} \leftarrow \frac{t^{(k)}-1}{t^{(k+1)}}$;
8: $y^{(k+1)} \leftarrow x^{(k)} + \beta^{(k)}(x^{(k+1)} - x^{(k)})$;
9: **return** $x^{(K)}$;

FISTA is not a monotone method as $F(x^{(k)})$ can be smaller than $F(x^{(k+1)})$. Beck and Teboulle [3] provide the following simple modification to the step 7 of Algorithm 3, which guarantees the objective function is non-increasing over iterations:

$$\beta^{(k)} = \begin{cases} \frac{t^{(k)}-1}{t^{(k+1)}}, & \text{if } F(x^{(k+1)}) < F(x^{(k)}), \\ 0, & \text{otherwise.} \end{cases}$$

This means that the extrapolation in the next iteration is temporarily disabled if the objective oscillates.

2.4 Summary and Remarks

The proximal method is also called generalized gradient descent, because the proximal operator can be considered as a projection. For a non-differentiable ℓ_1-regularized function, the proximal operator is a shrinkage operation, and it is called Iterative Shrinkage Threshold Algorithm (ISTA) [3].

In practice, accelerated first order methods and coordinate descent work better for solving sparse optimization problems [3, 28]. Thus when solving an optimization problem associated with a very large data set, coordinate descent is preferred over gradient descent methods. In particular, FISTA is considered one of the most competitive methods for solving the ℓ_1-regularized problems. Figure 2.1 shows the performances of ISTA and FISTA in solving the ℓ_1-regularized PageRank problem.

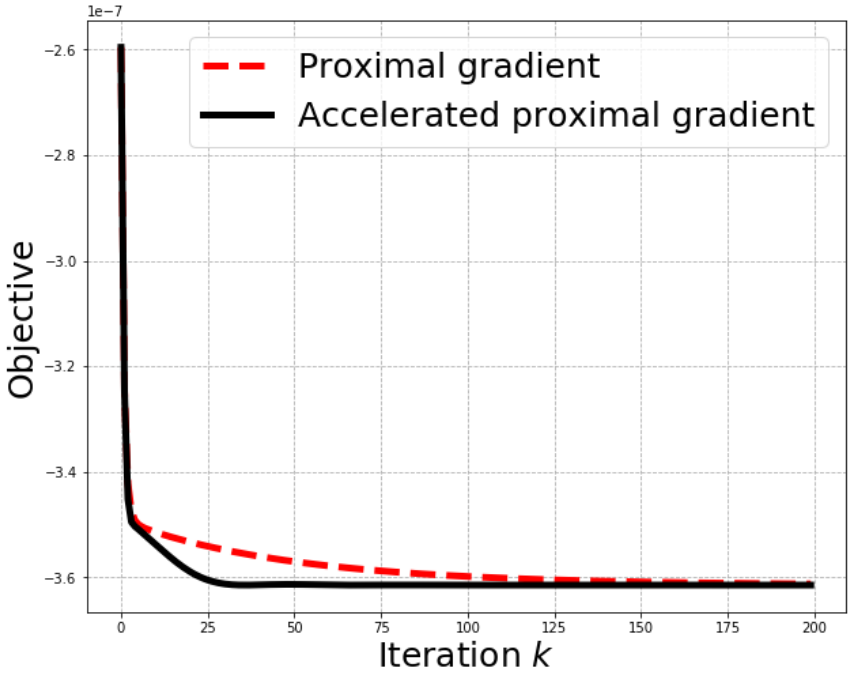

Figure 2.1: This plot shows the objective function values of proximal gradient and accelerated proximal gradient at each iteration when solving the ℓ_1-regularized PageRank problem (4.3). The data is a students' social network [42].

For a large data set, even computation of a function value is expensive. Moreover, data can be distributed in space and time. For such a problem, coordinate descent can be a good choice [28]. In Section 4.2, we propose a coordinate descent method to minimize an objective function with ℓ_1-regularization.

Chapter 3

Local Graph Clustering

Graph clustering algorithms answer the question that how the vertices should be grouped in a reasonable way. Intuitively, we want that there are a significant number of edges inside the groups and very few edges crossing the groups. Algorithms that eventually assign every vertex to at least one cluster are considered global methods, whereas in a local clustering, the cluster assignments are only done for a certain subset of vertices.

For large graphs, global clustering becomes computationally demanding and the running time often grows faster than $O(m)$ [36, 20, 31], where m is the number of vertices. Additional motivation for local graph clustering comes from large networks, where data is not explicitly available. For example, web page information is usually collected recursively by a web crawler.

Unfortunately, there is no uniform definition of a cluster in arbitrary graphs, and the variants used in clustering algorithms are developed for different demands of numerous applications [44]. The local graph clustering problem can be understood as a recovery problem. One assumes that there exists a target cluster in a given graph, and the objective is to recover the target cluster from one vertex inside the cluster [19].

In this study, we focus on the local graph clustering problem related to solving a convex optimization problem with a specific objective function (4.3). In Section 3.2, we will introduce the optimization problem formally.

3.1 PageRank in Local Graph Clustering

The core idea of PageRank is the random surfer walking on a graph of websites, where web pages are vertices and links are edges.

In a graph G of m vertices, the vector s represents a probability distribution of the starting vertices picked by the random surfer. For any $i \in [m]$ we have

$$s_i \begin{cases} > 0, & \text{if } i \text{ is a starting vertex.} \\ = 0, & \text{otherwise.} \end{cases}$$

Let A and D be the adjacency matrix and degree diagonal matrix of the graph. At vertex i, the random surfer transits to vertex j with probability $\frac{A_{ij}}{2d_i}$. Thus the lazy random walk transition matrix W is

$$W = \frac{1}{2}(I + D^{-1}A),$$

where I is the identity matrix.

3.1.1 Non-linear Random Walk

Starting from a web page (i.e. seed vertex), the surfer stays at the starting web page with probability $\alpha \in (0, 1]$, or teleports to other pages with probability $1 - \alpha$. Then at the k-th random walk step, we update the current distribution $q^{(k)}$ of the random surfer by

$$q^{(k+1)} = \alpha s + (1 - \alpha)Wq^{(k)}.$$

When $q^{(k+1)} = q^{(k)}$, we obtain a stationary distribution, which is the desired PageRank vector.

The idea of using PageRank for local graph clustering is that the seed vertices are in a cluster, and other vertices of that cluster are more likely to be visited by the random surfer. Therefore, the vertices with the highest probabilities in the stationary distribution should belong to the same cluster.

A problem of the random walk is that there will be small probabilities in vertices far away from the seed vertices. If we sort the stationary probability vector, we are likely to have a long tail. We want to get rid of the tail because we only care about vertices that contain a significant amount of probabilities. Thus we can set small values to zero with a threshold as follows

$$\tilde{q}_i^{(k)} = \begin{cases} q_i^{(k)} - \epsilon\alpha d_i, & \text{if } q_i^{(k)} > \epsilon\alpha d_i, \\ 0, & \text{otherwise,} \end{cases}$$

where ϵ is an approximate parameter as the result becomes an approximation to the PageRank vector. Use a point-wise max function, we obtain the non-linear transformation at the k-th step

$$\tilde{q}^{(k+1)} = \max\left\{\alpha s + (1-\alpha)W\tilde{q}^{(k)} - \epsilon\alpha d, \vec{0}\right\}. \tag{3.1}$$

In the work of Fountoulakis et al. [15], it is proved that the stationary obtained by taking an infinite sequence of non-linear random walk steps (3.1) is equivalent to the solution of the Approximate Personalized PageRank (APPR) algorithm.

3.1.2 Approximate the PageRank Vector

The PageRank vector $p(s) \in \mathbb{R}^m$ is a stationary probability distribution defined as the solution of the following linear system

$$p(s) = \alpha s + (1-\alpha)Wp(s). \tag{3.2}$$

Given the constant α, the PageRank vector is uniquely defined by s [2, 7], thus we denote it as $p(s)$.

APPR approximates the PageRank vector by another vector with a small residual. We present the definition of the approximate PageRank vector, which is first introduced in the work of Andersen et al [2].

Definition 3.1.1. *An ϵ-approximate PageRank vector for the linear system (3.2) is a PageRank vector $\widetilde{p}(\epsilon, s)$ for the linear system*

$$\widetilde{p}(\epsilon, s) = \alpha(s - r(\epsilon, s)) + (1-\alpha)W\widetilde{p}(\epsilon, s), \tag{3.3}$$

where the vector $r(\epsilon, s)$ is nonnegative and satisfies $r(\epsilon, s)_i \leq \epsilon d_i$ for every vertex i in the graph.

APPR (Algorithm 4) is introduced by Andersen et al. [2] to find $\widetilde{p}(\epsilon, s)$ and $r(\epsilon, s)$ defined in (3.3). Algorithm 4 initializes $p = \vec{0}$, and r as the vector identifying the seed vertices. In every iteration, the algorithm takes the probability from r at a single vertex i. Then it pushes an α fraction of this probability to p_i, and spreads the remaining $(1-\alpha)$ fraction within r. Note that the algorithm only does the push operations on vertices such that $r_i \geq \epsilon d_i$, which ensures a significant amount of probability will be moved. Finally, the returned p and r are $\widetilde{p}(\epsilon, s)$ and $r(\epsilon, s)$ for the ϵ-approximate PageRank.

Algorithm 4 APPR

procedure APPROXIMATEPR($s, \alpha, \epsilon, A, D$)
  $p \leftarrow \vec{0}$;
  $r \leftarrow s$;
  while there exists some vertex i such that $r_i \geq \epsilon d_i$ **do**
    Pick any vertex i such that $r_i \geq \epsilon d_i$;
    $p_i \leftarrow p_i + \alpha r_i$;
    $r_i \leftarrow \frac{(1-\alpha)r_i}{2}$;
    for every vertex j such that $i \sim j$ **do**
      $r_j \leftarrow r_j + \frac{(1-\alpha)r_i}{2d_j} A_{ij}$;
  return p **and** r;

In the work of Andersen et al. [2], it has been shown that Algorithm 4 has the following properties.

- Algorithm 4 returns an ϵ-approximate PageRank vector $\widetilde{p}(\epsilon, s)$ for any s and ϵ such that $\|s\|_1 \leq 1$ and $\epsilon \in (0, 1]$.
- The support of p defined in Algorithm 4 satisfies $vol(supp(p)) \leq \frac{2}{(1-\alpha)\epsilon}$.
- Algorithm 4 has time complexity of $O(\frac{1}{\epsilon\alpha})$.

Note that Algorithm 4 can be implemented by maintaining a *first-in, first-out* (FIFO) queue containing those vertices i satisfying $r_i \geq \epsilon d_i$. Since the size of the queue is bounded according to the second property, Algorithm 4 is a local method whose time complexity is independent of the size of the graph. In section 3.2, we will see that Algorithm 4 can be considered as a coordinate solver for a specific objective function.

3.1.3 Find a Cluster from the PageRank Vector

The sweep method has been widely used in spectral partitioning [39]. In a graph $G = (V, E)$, the edge boundary of a vertices set C is defined to be

$$\partial(C) := \big\{\{i, j\} | i \in C, j \notin C\big\}.$$

The conductance of a vertices set C is

$$\Phi(C) := \frac{|\partial(C)|}{\min\{vol(C), 2|E| - vol(C)\}},$$

which is widely used to measure the quality of the cluster identified by C [14]. Ideally, there should be a very few connections between clusters, thus a *good* clustering means the clusters have small conductances.

The sweep method produces a cut in the graph G from a PageRank vector $p(s)$. Assume the elements of $p(s)$ are sorted by $\frac{p(s)_i}{d_i}$ in descending order. Let M be the number of nonzero elements in $p(s)$, and denote the ordering of the elements as $\{v_1, v_2, \ldots, v_M\}$, thus we have

$$\frac{p(s)_{v_i}}{d_{v_i}} \geq \frac{p(s)_{v_j}}{d_{v_j}}, \forall i < j \leq M.$$

A sweep set, consisting of the first j vertices in $p(s)$, is defined as

$$C_j(p(s)) := \{v_1, v_2, \ldots, v_j\},\ j = 1, \ldots, M.$$

Define $\phi(p(s))$ as the sweep set with the smallest conductance

$$\phi(p(s)) := \min_{j \in [1,M]} \Phi(C_j(p(s))).$$

The cluster identified by the sweep set $\phi(p(s))$ is the desired result of the graph clustering method based on PageRank vector. Given the vector $p(s)$, finding $\phi(p(s))$ can be achieved by sorting $p(s)$, and iterating the nonzero elements and their neighborhood. Therefore, the time complexity of the sweep method is $O\left(vol(supp(p(s))) + M\log(M)\right)$.

To find a cut within a vertices set C, it suffices to compute an ϵ-approximate PageRank vector with ϵ roughly $O(\frac{1}{vol(C)})$ [2]. Combined with APPR, we have the following procedure: compute an ϵ-approximate PageRank vector $\widetilde{p}(\epsilon, s)$, then perform the sweep method over $\widetilde{p}(\epsilon, s)$. Andersen et al. [2] show that for any set C of conductance $O(\alpha)$, and for the seed vertices within C, this procedure finds a cluster with conductance $O(\sqrt{\alpha \log(vol(C))})$. In other words, this procedure can find a *sufficiently good* cluster around the seed vertices, which requires running time dependent on the volume of a subset in the graph. Figure 3.1 shows the local graph clustering result by performing APPR and sweep method, the procedure finds a single cluster around the seed vertex.

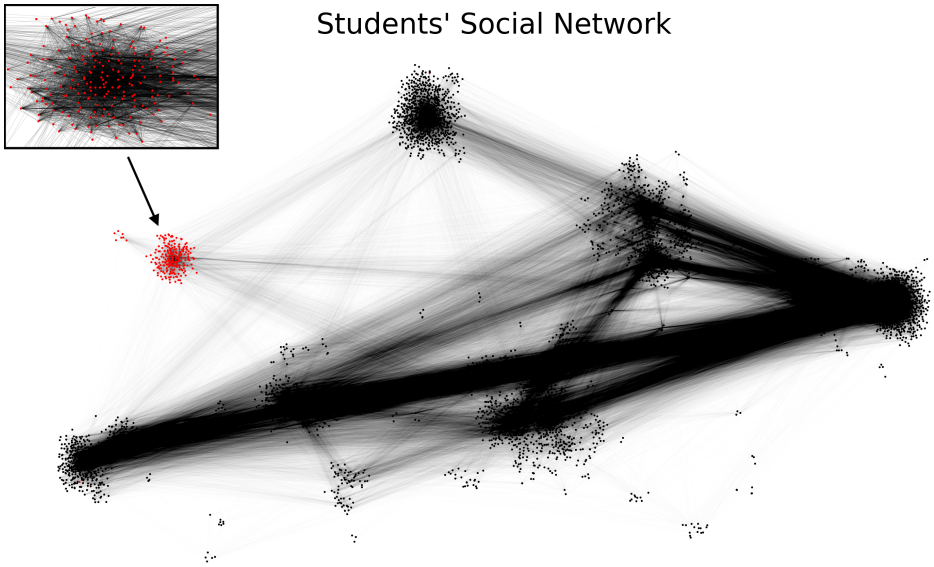

Figure 3.1: This figure shows the clustering result in a graph of students' social network [42]. The local procedure finds a cluster (i.e red points) around the seed vertex without touching the whole graph.

3.2 PageRank: an Optimization View

This section briefly reviews the connection between the PageRank problem and an optimization framework. Let $p(s)$ be the unique solution of the linear system (3.2). The simplest way to solve this linear system is updating the vector $p(s)$ iteratively until the equation (3.2) holds true [16]. Another way of solving the PageRank problem is minimizing the Euclidean norm of the residual of (3.2) as following

$$\min_{p} \|(I - (1-\alpha)W)p - \alpha s\|_2.$$

When the distance is 0, we get the desired PageRank vector $p(s)$. For an approximate algorithm, it is often enough to find an approximation $\widetilde{p}$ such that the distance is smaller than a threshold. Mathematically, we can write it as

$$\|(I - (1-\alpha)W)\widetilde{p} - \alpha s\|_2 \leq \eta,$$

where $\eta > 0$ is a constant. This expression means that if the distance is small enough, the approximate algorithm stops earlier than algorithms finding the exact solution. This

termination criteria gives us a hint that APPR might be similar to optimization methods, since $\|\nabla F(x)\|_2 \leq \eta$ is commonly used as a termination criteria in many algorithms that minimize a differentiable function F.

In fact, Fountoulakis et al. [15] show that APPR (i.e Algorithm 4) is an iterative coordinate solver for the linear system (3.2). Moreover, Algorithm 4 minimizes the following objective function f,

$$f(q) := \frac{1}{2}\langle q, Qq\rangle - \alpha\langle s, D^{\frac{1}{2}}q\rangle, \tag{3.4}$$

where the symmetric positive definite matrix Q is defined as

$$Q := D^{-\frac{1}{2}}\left\{D - \frac{1-\alpha}{2}(D+A)\right\}D^{-\frac{1}{2}}.$$

Figure 3.2 demonstrates the objective function value and number of nonzero entries in p at each iteration of Algorithm 4 for a sample problem.

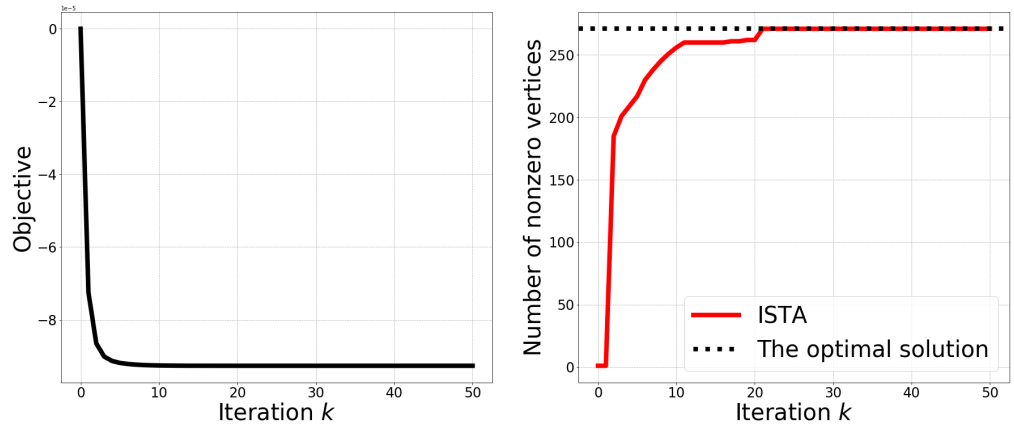

Figure 3.2: This figure shows the results of Algorithm 4 in the graph of web pages. The optimal solution is the solution of (4.3). The data was release in 2002 by Google as a part of Google Programming Contest.

Chapter 4

Optimization Methods for ℓ_1-regularized PageRank Problem

In this chapter, we present the proximal gradient descent and coordinate descent for solving the ℓ_1-regularized PageRank problem. We also prove the random coordinate method is a local method and derive its iteration complexity.

In Section 3.2, we have shown that one can find the desired PageRank vector by minimizing the following objective function $f : \mathbb{R}^m \to \mathbb{R}$

$$f(q) := \frac{1}{2}\langle q, Qq\rangle - \alpha\langle s, D^{\frac{1}{2}}q\rangle, \tag{4.1}$$

where Q is defined as

$$Q := D^{-\frac{1}{2}}\left\{D - \frac{1-\alpha}{2}(D+A)\right\}D^{-\frac{1}{2}}. \tag{4.2}$$

One advantage of this interpretation is that we can compute the PageRank vector through various optimization algorithms. In the work of Fountoulakis et al [15], it is proved that an ϵ-approximate PageRank vector is indeed the solution of an ℓ_1-regularized problem

$$\min_q F(q) := \rho\alpha\|D^{\frac{1}{2}}q\|_1 + f(q), \tag{4.3}$$

where $f(q)$ is defined in (4.1), and ρ is the approximate parameter in Algorithm 4.

4.1 Proximal Gradient Descent

Since the function $F(q)$ in (4.3) has the ℓ_1 regularization, one can use proximal gradient descent (i.e. Algorithm 5) to minimize it. Additionally, Fountoulakis et al. [15] demonstrate that Algorithm 5 has the following important properties

- For any coordinate i, we have $0 \leq q_i^{(k)} \leq q_i^{(k+1)}\ \forall k = 0, 1 \ldots$.
- Define $S^{(k)} := \{i | q_i^{(k)} - \nabla f_i(q^{(k)}) \geq \rho\alpha d_i^{\frac{1}{2}}\}$, then $S^{(k)} \subseteq S^{(k+1)}\ \forall k = 0, 1 \ldots$.
- $\nabla f(q^{(k)}) \leq 0\ \forall k = 0, 1 \ldots$.
- $\nabla_i f(q^{(k)}) \leq -\rho\alpha d_i^{\frac{1}{2}}\ \forall i \in S^{(k)}$.

These properties are used to show that the step 5 of Algorithm 5 can be simplified to

$$q_i^{(k+1)} = \begin{cases} q_i^{(k)} - (\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}) & \text{if } i \in S^{(k)}, \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 5 ISTA for solving (4.3)

1: **procedure** MINIMIZE(ρ, α, s, D)
2: $q_0 \leftarrow \vec{0}$;
3: $\nabla f(q_0) \leftarrow -\alpha D^{-\frac{1}{2}} s$;
4: **for** $k = 0, 1, \ldots, K-1$ **do**
5:

$$q_i^{(k+1)} = \begin{cases} q_i^{(k)} - (\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}) & \text{if } q_i^{(k)} - \nabla f_i(q^{(k)}) \geq \rho\alpha d_i^{\frac{1}{2}}, \\ q_i^{(k)} - (\nabla_i f(q^{(k)}) - \rho\alpha d_i^{\frac{1}{2}}) & \text{if } q_i^{(k)} - \nabla f_i(q^{(k)}) \leq -\rho\alpha d_i^{\frac{1}{2}}, \\ 0 & \text{otherwise.} \end{cases}$$

6: Compute $\nabla f(q^{(k+1)})$;
7: **return** $p^{(K)} := D^{\frac{1}{2}} q^{(K)}$;

Let q^* be the optimal solution of (4.3), and $\mathcal{F}^* := supp(q^*)$ be the support of the solution. Another important property of Algorithm 5 is that the volume of $S^{(k)}$ is upper bounded by $\frac{\|s\|_1}{\rho}$ [15], where s is the probability distribution of the seed vertices. Since

$e^T s = 1$ and $s \geq 0$, the number of vertices visited by Algorithm 5 in every iteration is in the order of $O(\frac{1}{\rho})$.

Figure 4.1 shows the local clustering result obtained by Algorithm 5 and the sweep method defined in Section 3.1.3. In Chapter 6, we will show that the boundary and scale of the detected cluster are mainly controlled by the regularization parameter ρ and teleport probability α.

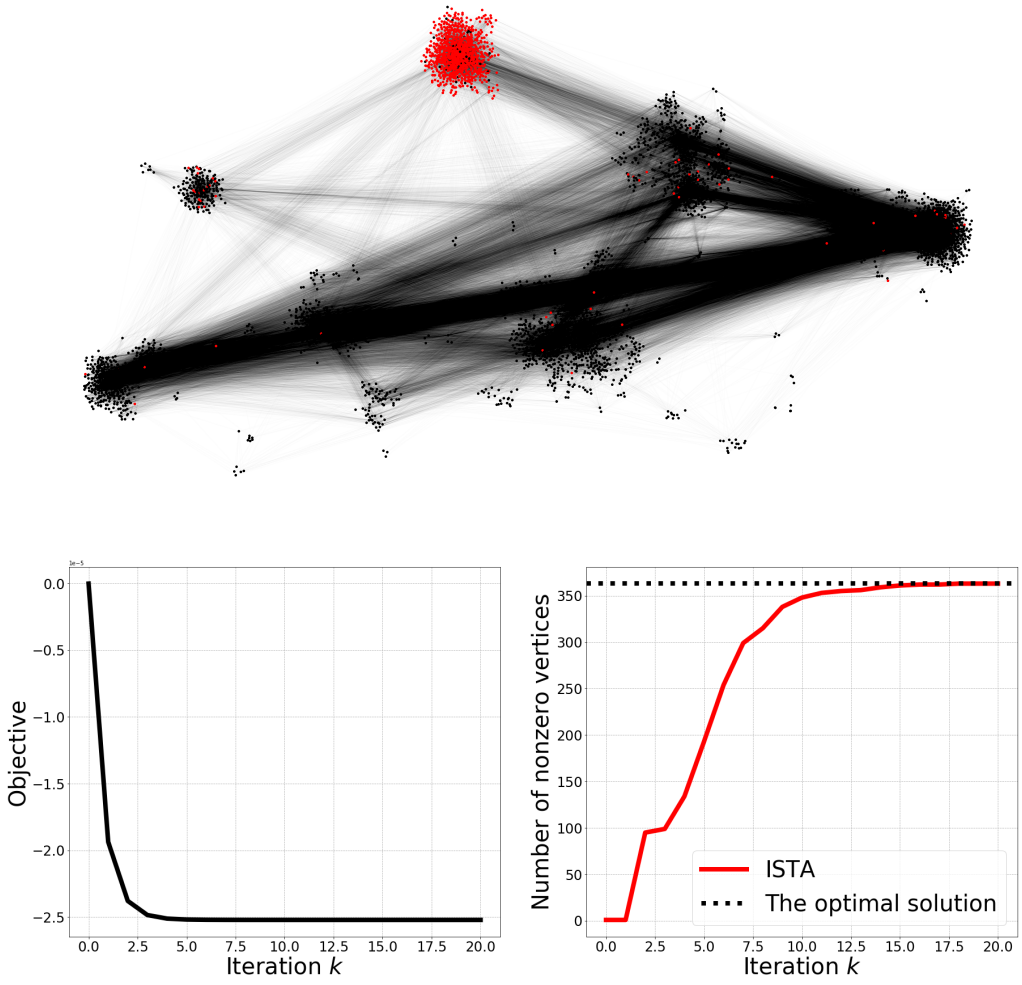

Figure 4.1: The top plot shows the sweep method finds a cluster (i.e. red points) from the PageRank vector. The bottom plots show the function value and the number of nonzero vertices in each iteration of the Algorithm 5 for a sample problem. The optimal solution is the solution of (4.3). The graph is a students' social network [42].

4.2 Random Coordinate Minimization

The idea behind coordinate descent is minimizing the objective function by successively processing each coordinate of the function in a cyclic or random fashion. An advantage of using coordinate descent is that when the objective function f is quadratic, coordinate methods exactly minimize the function over the given coordinate direction. In the k-th iteration, coordinate methods solve the single variable sub-problem for the i-th coordinate as follows

$$q_i^{(k+1)} = \underset{q_i}{\operatorname{argmin}} f(q_1^{(k)}, \ldots, q_{i-1}^{(k)}, q_i, q_{i+1}^{(k)}, \ldots, q_m^{(k)}).$$

When the sub-problems can be solved quickly, coordinate methods are very efficient [13, 28, 34].

4.2.1 Coordinate-wise Lipschitz Continuity

Let $f : \mathbb{R}^m \to \mathbb{R}$ be the function defined in (4.1). The coordinate Lipschitz constant L_i of f is defined as

$$|\nabla_i f(q + te(i)) - \nabla_i f(q)| \leq L_i |t|, \tag{4.4}$$

With the mean value theorem, we get L_i by deriving the upper bound of $|\nabla_i^2 f(q)|$. The second order partial derivative of f is

$$\begin{aligned} \nabla_i^2 f(q) &\overset{(4.1)}{=} Q_{ii} \\ &\overset{(4.2)}{=} \frac{1+\alpha}{2} + \frac{A_{ii}}{d_i} \\ &= \frac{1+\alpha}{2}. \end{aligned}$$

The last equality is because we assume that graphs have no self-loops (i.e. $A_{ii} = 0$, $\forall i$). An important consequence of (4.4) is the following standard inequality [30] for any $t \in \mathbb{R}$:

$$f(q + e(i)t) \leq f(q) + \langle \nabla_i f(q), t \rangle + \frac{L_i}{2} \|t\|_2^2. \tag{4.5}$$

For simplicity, we define the function $V_i(q, t)$ as

$$V_i(q, t) := \langle \nabla_i f(q), t \rangle + \frac{L_i}{2} \|t\|_2^2 + \rho \alpha d_i^{\frac{1}{2}} |q_i + t|, \tag{4.6}$$

which can be considered as a coordinate wise approximation function similar to (2.2). Let Pr be a probability distribution, and we pick the vertex i with probability Pr_i. Now we are ready to present the generalized randomized coordinate descent for minimizing the function $F : \mathbb{R}^m \to \mathbb{R}$ defined in (4.3) in Algorithm 6, which is first introduced in the work of Richtárik et al. [34]. Compared with Algorithm 2, Algorithm 6 performs a *coordinate wise* proximal step at every iteration.

Algorithm 6 Generic randomized coordinate descent for solving (4.3)

procedure MINIMIZE(Pr, $q^{(0)}$)
  for $k = 0, 1, 2, \ldots, K-1$ **do**
    Choose $i \in \{1, 2, \ldots, m\}$ with probability Pr_i;
    $\Delta q_i^{(k)} \leftarrow \operatorname{argmin}_t V_i(q^{(k)}, t)$;
    $q^{(k+1)} \leftarrow q^{(k)} + e(i)\Delta q_i^{(k)}$;
  return $q^{(K)}$;

4.2.2 Local Properties of the Coordinate Descent

In the work of Fountoulakis et al. [15], it has been proved that proximal gradient descent is a local method. Similarly, we derive the local properties of the coordinate descent method for solving the ℓ_1-regularized PageRank problem (4.3).

Since we focus on a subset of the graph in local clustering problems, we make an assumption to ensure that there is significant probability in a few seed vertices [2, 15]. For any seed vertex i, we have

$$s_i \geq \rho d_i, \tag{4.7}$$

where s is probability distribution of seed vertices, and ρ is the regularization parameter in (4.3).

To find the solution of $\operatorname{argmin}_t V_i(q^{(k)}, t)$ defined in (4.6), we consider the following three different cases. When $t + q_i^{(k)} > 0$, we have

$$V_i(q^{(k)}, t) = \langle \nabla_i f(q^{(k)}), t \rangle + \frac{L_i}{2}\|t\|_2^2 + \rho\alpha d_i^{\frac{1}{2}}(q_i^{(k)} + t). \tag{4.8}$$

Since (4.8) is convex, we find the minimum point by setting its derivative to zero

$$\nabla_i f(q^{(k)}) + L_i t + \rho\alpha d_i^{\frac{1}{2}} = 0,$$

which indicates $t = -\frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right)$ and $q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) \geq \frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}}$.

When $t + q_i^{(k)} < 0$, we have

$$V_i(q^{(k)}, t) = \langle \nabla_i f(q^{(k)}), t\rangle + \frac{L_i}{2}\|t\|_2^2 - \rho\alpha d_i^{\frac{1}{2}}(q_i^{(k)} + t). \tag{4.9}$$

Since (4.9) is convex, we find the minimum point by setting its derivative to zero

$$\nabla_i f(q^{(k)}) + L_i t - \rho\alpha d_i^{\frac{1}{2}} = 0,$$

which indicates $t = -\frac{1}{L_i}\left(\nabla_i f(q^{(k)}) - \rho\alpha d_i^{\frac{1}{2}}\right)$ and $q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) \leq -\frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}}$.

When $t + q_i^{(k)} = 0$, we have

$$V_i(q^{(k)}, t) = \langle \nabla_i f(q^{(k)}), t\rangle + \frac{L_i}{2}\|t\|_2^2. \tag{4.10}$$

Since (4.10) is convex, we find the minimum point by setting its derivative to zero

$$\nabla_i f(q^{(k)}) + L_i t = 0,$$

which indicates $t = -\frac{\nabla_i f(q^{(k)})}{L}$ and $-\frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}} < q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) < \frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}}$. To summarize these three cases, we define their corresponding sets at iteration k of Algorithm 6 as follows

$$\begin{aligned}
S^{(k)} &:= \left\{ i \in [m] \middle| q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) \geq \frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}} \right\}, \\
\widetilde{S}^{(k)} &:= \left\{ i \in [m] \middle| q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) \leq -\frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}} \right\}, \\
\widehat{S}^{(k)} &:= \left\{ i \in [m] \middle| -\frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}} < q_i^{(k)} - \frac{1}{L_i}\nabla_i f(q^{(k)}) < \frac{1}{L_i}\rho\alpha d_i^{\frac{1}{2}} \right\}.
\end{aligned} \tag{4.11}$$

The solution of $\operatorname{argmin}_t V_i(q^{(k)}, t)$ implies that the step 4 and 5 in Algorithm 6 is equivalent to

$$q_i^{(k+1)} = \begin{cases} q_i^{(k)} - \frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right), & \text{if } i \in S^{(k)}, \\ q_i^{(k)} - \frac{1}{L_i}\left(\nabla_i f(q^{(k)}) - \rho\alpha d_i^{\frac{1}{2}}\right), & \text{if } i \in \widetilde{S}^{(k)}, \\ 0, & \text{if } i \in \widehat{S}^{(k)}. \end{cases} \tag{4.12}$$

We introduce the following theorem to show the local properties of Algorithm 6.

Theorem 1. *Let q^* be the optimal solution of problem (4.3), and $\rho, \alpha > 0$. Let f be the function defined in (3.4), and $\{q^{(k)}\}$ be the sequence generated by Algorithm 6. Use the sets defined in (4.11). Algorithm 6, assuming $q^{(0)} = \vec{0}$, has the following properties at any iteration $k = 0, 1, \ldots$.*

1. $\tilde{S}^{(k)} = \emptyset$.
2. $q_i^{(k)} = 0$ *and* $\nabla_i f(q^{(k)}) \leq 0 \ \forall i \notin S^{(k)}$.
3. $q_i^{(k)} \geq 0$ *and* $\nabla_i f(q^{(k)}) \leq -\rho\alpha d_i^{\frac{1}{2}} \ \forall i \in S^{(k)}$.
4. $S^{(k)} \subseteq S^{(k+1)} \subseteq supp(q^*)$.

Proof. Now we use induction to prove Theorem 1. The gradient of f is

$$\nabla f(q) = Qq - \alpha D^{-\frac{1}{2}} s, \tag{4.13}$$

where $s \geq 0$ and $e^T s = 1$.

When $k = 0$, since $q^{(0)} = \vec{0}$, the gradient is

$$\nabla_i f(q^{(0)}) = \begin{cases} -\alpha s_i d_i^{-\frac{1}{2}} < 0, & \text{if } s_i \neq 0, \\ 0, & \text{if } s_i = 0, \end{cases} \quad i \in [m].$$

Combined with the assumption (4.7), it is easy to see that S_0 is the set of seed vertices and the first 3 properties hold true when $k = 0$.

When $k > 0$, we assume that the first 3 properties hold true, we prove that they are still valid at iteration $k + 1$. Since $\widetilde{S}^{(k)} = \emptyset$, (4.12) can be simplified to

$$q_i^{(k+1)} = \begin{cases} q_i^{(k)} - \frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right), & \text{if } i \in S^{(k)}, \\ 0, & \text{if } i \notin S^{(k)}. \end{cases} \tag{4.14}$$

With the assumption that $\nabla_i f(q^{(k)}) \leq -\rho\alpha d_i^{\frac{1}{2}}$ and $q_i^{(k)} \geq 0 \ \forall i \in S^{(k)}$, we have

$$q_i^{(k+1)} \begin{cases} \geq q_i^{(k)} \geq 0, & \text{if } i \in S^{(k)}, \\ = 0, & \text{if } i \notin S^{(k)}. \end{cases} \tag{4.15}$$

For simplicity, we define $\Delta q_i^{(k)}$ as follows

$$\begin{aligned}\Delta q_i^{(k)} &= q_i^{(k+1)} - q_i^{(k)} \\ &= \begin{cases} -\frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right) \geq 0, & \text{if } i \in S^{(k)}, \\ 0, & \text{if } i \notin S^{(k)}. \end{cases}\end{aligned}$$

When Algorithm 6 picks coordinate $i \notin S^{(k)}$ at iteration k, it is obvious that $q^{(k)} = q^{(k+1)}$ and $S^{(k)} = S^{(k+1)}$. Since nothing changes at this iteration, all properties are still valid at iteration $k+1$.

When Algorithm 6 picks coordinate $i \in S^{(k)}$ at iteration k. The gradient at iteration $k+1$ can be computed in the following way

$$\nabla_j f(q^{(k+1)}) = \begin{cases} -\rho\alpha d_j^{\frac{1}{2}} - \frac{1-\alpha}{2}\Delta q_j^{(k)}, & \text{if } j = i, \\ \nabla_j f(q^{(k)}) - \frac{(1-\alpha)A_{ij}}{2d_j^{\frac{1}{2}} d_i^{\frac{1}{2}}}\Delta q_i^{(k)}, & \text{if } j \sim i, \\ \nabla_j f(q^{(k)}), & \text{otherwise.} \end{cases} \tag{4.16}$$

Equation (4.16) is because we have

$$\nabla f(q^{(k+1)}) = \nabla f(q^{(k)}) + \Delta q - \frac{1-\alpha}{2}\Delta q - \frac{1-\alpha}{2} D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \Delta q,$$

where $\Delta q := q^{(k+1)} - q^{(k)}$. Since $\Delta q_i^{(k)} \geq 0$ and we have assumed that $\nabla_i f(q^{(k)}) \leq 0$ $\forall i \in [m]$, (4.16) implies that

$$\nabla_j f(q^{(k+1)}) \begin{cases} \leq -\rho\alpha d_j^{\frac{1}{2}} < 0, & \text{if } j = i, \\ \leq \nabla_j f(q^{(k)}) \leq 0, & \text{if } j \sim i, \\ = \nabla_j f(q^{(k)}) \leq 0, & \text{otherwise.} \end{cases} \tag{4.17}$$

By combining (4.11), (4.17) and (4.15), we can make the following conclusions

- For any coordinates $j \in [m]$, we always have $q_j^{(k+1)} \geq 0$ and $\nabla_j f(q^{(k+1)}) \leq 0$, thus $q_j^{(k+1)} - \frac{1}{L_j}\nabla_j f(q^{(k+1)}) \geq 0$. As a result, we get $\widetilde{S}^{(k+1)} = \emptyset$.

- For any coordinate $j \in S^{(k)}$, because $\nabla_j f(q^{(k+1)}) \leq -\rho\alpha d_j^{\frac{1}{2}}$ and $q_j^{(k+1)} \geq q_j^{(k)}$ we must have $j \in S^{(k+1)}$, thus $S^{(k)} \subseteq S^{(k+1)}$. Using the fact that Algorithm 6 converges [34], we have $S^{(k)} \subseteq S^{(k+1)} \subseteq supp(q^*)$.

- For any coordinate $j \notin S^{(k)}$ but will be in $S^{(k+1)}$, because $q_j^{(k+1)} = 0$ and $\nabla_j f(q^{(k+1)}) \leq 0$ we must have $\nabla_j f(q^{(k+1)}) \leq -\rho\alpha d_j^{\frac{1}{2}}$.

Hence all properties hold true at iteration $k+1$. □

Fountoulakis et al. [15] proves that $vol(supp(q^*))$ is upper bounded by $\frac{\|s\|_1}{\rho}$, which indicates that the number of nonzero entries in $q^{(k)}$ of Algorithm 6 is upper bounded and independent of the size of the graph. Therefore, Theorem 1 shows the locality of Algorithm 6.

An observation of Theorem 1 is that Algorithm 6 updates $q^{(k)}$ only if it picks coordinates from $S^{(k)}$. To see this, we can rewrite step 4 and 5 of Algorithm 6 as

$$q_i^{(k+1)} = \begin{cases} q_i^{(k)} - \frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right), & \text{if } i \in S^{(k)}, \\ 0, & \text{if } i \in \widehat{S}^{(k)}, \end{cases}$$

where $\widetilde{S}^{(k)}$ is ignored because of the first property of Theorem 1. In other words, step 4 of Algorithm 6 performs in the following way

$$\operatorname*{argmin}_t V_i(q^{(k)}, t) = \begin{cases} -\frac{1}{L_i}\left(\nabla_i f(q^{(k)}) + \rho\alpha d_i^{\frac{1}{2}}\right), & \text{if } i \in S^{(k)}, \\ 0, & \text{if } i \notin S^{(k)}. \end{cases} \tag{4.18}$$

Therefore, we should let Algorithm 6 only picks coordinates from $S^{(k)}$. Algorithm 7 shows the modified version. The probability distribution of picking coordinates is defined over all nodes, but for coordinates outside of $S^{(k)}$ the probability is 0, and coordinates in $S^{(k)}$ are picked with the same probability.

Algorithm 7 Random coordinate descent for solving (4.3)

1: **procedure** MINIMIZE($q^{(0)}$)
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Choose $i \in S^{(k)}$ with probability $\frac{1}{|S^{(k)}|}$; ▷ $S^{(k)}$ defined in (4.11).
4: $\Delta q_i^{(k)} \leftarrow \operatorname{argmin}_t V_i(q^{(k)}, t)$;
5: $q^{(k+1)} \leftarrow q^{(k)} + e(i)\Delta q_i^{(k)}$;
6: **return** $q^{(K)}$;

4.2.3 Complexity Analysis

In this section, we derive the iteration complexity of Algorithm 7, which is the main contribution of this thesis. We define the following function $H(q, T)$ that plays an important role in this analysis

$$H(q,T) := f(q) + \langle \nabla f(x), T\rangle + \frac{1}{2}\|T\|_L^2 + \rho\alpha\|D^{\frac{1}{2}}(q+T)\|_1 \quad q, T \in \mathbb{R}^m, \tag{4.19}$$

where $f(q)$ is defined in (4.1), and $\|T\|_L$ is defined with the coordinate wise Lipschitz constants of f:

$$\|T\|_L := \sqrt{\sum_{i=1}^{m} L_i |T_i|^2}.$$

Use the definition of $V_i(q,t)$ in (4.6), $H(q,T)$ can be written as

$$H(q,T) = f(q) + \sum_{i=1}^{m} V_i(q, T_i). \tag{4.20}$$

It is easy to see that the vector $\Delta q^{(k)} = (\Delta q^{(k)}(1), \Delta q^{(k)}(2), \ldots, \Delta q^{(k)}(m))$, with the $\Delta q_i^{(k)}$ defined in Algorithm 7 line 4, is the minimizer of $H(q^{(k)}, T)$:

$$\Delta q^{(k)} = \operatorname*{argmin}_{T\in\mathbb{R}^m} H(q^{(k)}, T).$$

We derive Lemma 2 that will be used in later analysis

Lemma 2. *Let $\{q^{(k)}\}$ be the random iterates generated by Algorithm 7 and $S^{(k)}$ be the set defined in (4.11). Consider the function $F(q)$ defined in (4.3) and denote q^* as the point minimizes F. Then*

$$\mathbb{E}[F(q^{(k+1)}) - F(q^*)|q^{(k)}] \le \frac{1}{|S^{(k)}|}(H(q^{(k)}, \Delta q^{(k)}) - F(q^*)) + \frac{|S^{(k)}|-1}{|S^{(k)}|}(F(q^{(k)}) - F(q^*)). \tag{4.21}$$

Proof. Define the following function

$$C_i(q) := \rho\alpha \sum_{j\neq i} d_j^{\frac{1}{2}} |q_j|. \tag{4.22}$$

For any $q \in \mathbb{R}^m$, we have

$$\begin{aligned} F(q + e(i)t) &= f(q + e(i)t) + \rho\alpha \|D^{\frac{1}{2}}(q + e(i)t)\|_1 \\ &\overset{(4.5),(4.6)}{\leq} f(q) + V_i(q, t) + C_i(q). \end{aligned} \tag{4.23}$$

Then we get

$$\begin{aligned} \mathbb{E}[F(q^{(k+1)})|q^{(k)}] &= \frac{1}{|S^{(k)}|} \sum_{i \in S^{(k)}} F(q^{(k)} + e(i)\Delta q_i^{(k)}) \\ &\overset{(4.23)}{\leq} \frac{1}{|S^{(k)}|} \sum_{i \in S^{(k)}} [f(q^{(k)}) + V_i(q^{(k)}, \Delta q_i^{(k)}) + C_i(q^{(k)})] \end{aligned} \tag{4.24}$$

Combine Theorem 1 and (4.24) to get

$$\begin{aligned} &\frac{1}{|S^{(k)}|} \sum_{i \in S^{(k)}} [f(q^{(k)}) + V_i(q^{(k)}, \Delta q_i^{(k)}) + C_i(q^{(k)})] \\ &\overset{(4.20)}{=} \frac{1}{|S^{(k)}|} H(q^{(k)}, \Delta q^{(k)}) + \frac{|S^{(k)}| - 1}{|S^{(k)}|} f(q^{(k)}) + \frac{1}{|S^{(k)}|} \sum_{i \in S^{(k)}} C_i(q^{(k)}) \\ &\overset{(4.22)}{=} \frac{1}{|S^{(k)}|} H(q^{(k)}, \Delta q^{(k)}) + \frac{|S^{(k)}| - 1}{|S^{(k)}|} f(q^{(k)}) + \frac{1}{|S^{(k)}|} \sum_{i \in S^{(k)}} \sum_{j \neq i} \rho\alpha d_j^{\frac{1}{2}} |q_j^{(k)}| \\ &\overset{(4.3)}{=} \frac{1}{|S^{(k)}|} H(q^{(k)}, \Delta q^{(k)}) + \frac{|S^{(k)}| - 1}{|S^{(k)}|} F(q^{(k)}), \end{aligned}$$

where the first and second equalities are because Theorem 1 shows $V_i(q^{(k)}, \Delta q_i^{(k)}) = 0$ and $q_i^{(k)} = 0$ for any $i \notin S^{(k)}$. □

From Lemma 2, we see that the expected difference of $F(q^{(k+1)})$ from $F(q^*)$ is upper bounded by the function $H(q^{(k)}, \Delta q^{(k)})$. Since the function F defined in (4.3) is α-strongly convex [15], we get

$$F(x) \geq F(q) + \langle F'(q), x - q \rangle + \frac{\alpha}{2}\|x - q\|_2^2, \tag{4.25}$$

where $x, q \in \mathbf{dom}\, F$ and $F'(q)$ is the subgradient of F. Additionally, we have $\alpha \leq 1$ and $L_i = \frac{1+\alpha}{2} \leq 1 \; \forall i \in [m]$. Therefore, it is easy to see that F is α-strongly convex with respect to the $\|\cdot\|_L$ norm, that is

$$F(x) \geq F(q) + \langle F'(q), x - q \rangle + \frac{\alpha}{2}\|x - q\|_L^2. \tag{4.26}$$

Let q^* be the point that minimizes F. From the first order optimality condition for the problem (4.3), we obtain $\langle F'(q^*), q - q^* \rangle \geq 0$ for all $q \in \mathbf{dom}\ F$, which combining with (4.26), yields the standard inequality

$$F(q) - F(q^*) \geq \frac{\alpha}{2}\|q - q^*\|_L^2,\ \forall q \in \mathbf{dom}\ F. \tag{4.27}$$

We introduce the following lemma that will be useful to derive the convergence of the strongly convex function.

Lemma 3. *Let q^* be the solution of (4.3), and $\Delta q = \operatorname{argmin}_{T\in\mathbb{R}^m} H(q,T)$, where the function H is defined in (4.19). Let F be the function defined in (4.3) that has strongly convexity parameter $\alpha \in (0,1]$ with respect to the norm $\|\cdot\|_L$, then*

$$H(q,\Delta q) - F(q^*) \leq (1 - \frac{\alpha}{4})(F(q) - F(q^*)),\ q \in \boldsymbol{dom}\, F. \tag{4.28}$$

Proof.

$$\begin{aligned} H(q,\Delta q) &= \min_{T\in\mathbb{R}^m} H(q,T) \\ &= \min_{x\in\mathbb{R}^m} H(q, x-q) \\ &\overset{(4.19)}{=} \min_{x\in\mathbb{R}^m} f(q) + \langle \nabla f(q), x-q\rangle + \rho\alpha\|D^{\frac{1}{2}}x\|_1 + \frac{1}{2}\|x-q\|_L^2 \\ &\leq \min_{x\in\mathbb{R}^m} F(x) + \frac{1}{2}\|x-q\|_L^2, \end{aligned} \tag{4.29}$$

where the last inequality is because $f(q)$ defined in (4.1) is convex. Denote the point between q^* and q as $\delta q^* + (1-\delta)q$, $\delta \in [0,1]$, then we have

$$\begin{aligned} \min_{x\in\mathbb{R}^m} F(x) + \frac{1}{2}\|x-q\|_L^2 &\leq \min_{\delta\in[0,1]} F(\delta q^* + (1-\delta)q) + \frac{\delta^2}{2}\|q-q^*\|_L^2 \\ &\leq \min_{\delta\in[0,1]} F(q) - \delta(F(q) - F(q^*)) + \frac{\delta^2}{2}\|q-q^*\|_L^2 \\ &\overset{(4.27)}{\leq} \min_{\delta\in[0,1]} F(q) + \delta\left(\frac{\delta}{\alpha} - 1\right)(F(q) - F(q^*)). \end{aligned} \tag{4.30}$$

where the second inequality is because $F(q)$ defined in (4.3) is convex. Minimize the right hand side of (4.30) in δ to get $\delta^* = \frac{\alpha}{2}$. □

Finally, we derive Theorem 4 to show the expected value of $F(q^{(k)})$ converges to $F(q^*)$ linearly.

Theorem 4. *Let F be the function defined in (4.3) that has strongly convexity parameter $\alpha \in (0, 1]$ with respect to the norm $\|\cdot\|_L$. Let q^* be the point that minimizes F. If $\{q^{(k)}\}$ is the random sequence generated by Algorithm 7, then*

$$\mathbb{E}[F(q^{(k)}) - F(q^*)] \leq \left(1 - \frac{\alpha}{4|supp(q^*)|}\right)^k (F(q^{(0)}) - F(q^*)).$$

Proof. Combine Lemma 2 and Lemma 3 to get

$$\begin{aligned}
\mathbb{E}[F(q^{(k+1)}) - F(q^*)|q^{(k)}] &\overset{(4.21)}{\leq} \frac{1}{|S^{(k)}|}(H(q^{(k)}, \Delta q^{(k)}) - F(q^*)) + \frac{|S^{(k)}| - 1}{|S^{(k)}|}(F(q^{(k)}) - F(q^*)) \\
&\overset{(4.28)}{\leq} \frac{1}{|S^{(k)}|}\left(1 - \frac{\alpha}{4}\right)(F(q^{(k)}) - F(q^*)) + \frac{|S^{(k)}| - 1}{|S^{(k)}|}(F(q^{(k)}) - F(q^*)) \\
&= \left(1 - \frac{\alpha}{4|S^{(k)}|}\right)(F(q^{(k)}) - F(q^*)). \qquad (4.31)
\end{aligned}$$

Take expectation on both sides of (4.31) and apply it recursively to get

$$\begin{aligned}
\mathbb{E}[F(q^{(k)}) - F(q^*)] &\leq \prod_{i=0}^{k-1}\left(1 - \frac{\alpha}{4|S^{(i)}|}\right)(F(q^{(0)}) - F(q^*)) \\
&\leq \left(1 - \frac{\alpha}{4|supp(q^*)|}\right)^k (F(q^{(0)}) - F(q^*)).
\end{aligned}$$

In the last inequality, we use the result of Theorem 1, which indicates that $|S^{(k)}| \leq |supp(q^*)|$ $\forall k$. □

4.3 Summary and Remarks

Let q^* be the optimal solution of the ℓ_1-regularized PageRank problem (4.3). In Theorem 1, we have shown that $S^{(k)} \subseteq supp(q^*)$ $\forall k$, which implies that $|S^{(k)}| \leq |supp(q^*)|$. Therefore, the convergence rate, derived in Theorem 4, depends on the number of nonzero vertices of the optimal solution instead of the size of the whole graph. Thus the random coordinate descent is a local method. This gives an important advantage to randomized methods for problems of sufficiently large size. Figure 4.2 shows an example of objective function values at each iteration. Figure 4.3 shows the number of nonzero vertices in Algorithm 7 is upper bounded by $|supp(q^*)|$.

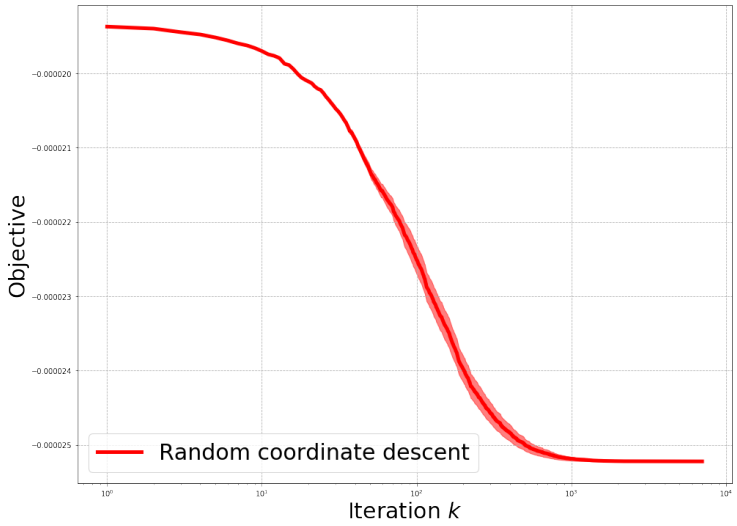

Figure 4.2: This figure shows the function value in each iteration of Algorithm 7 for solving the ℓ_1-regularized PageRank problem (4.3). The graph is a students' social network [42].

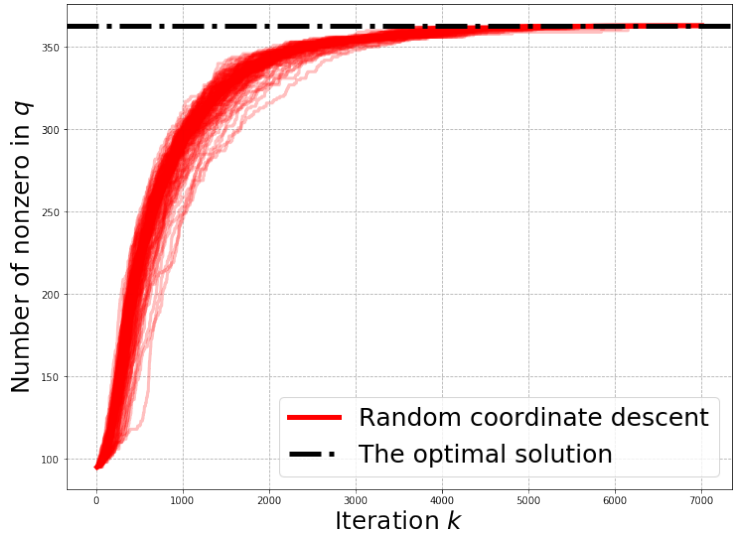

Figure 4.3: This figure shows the results of multiple trials of random coordinate descent. The optimal solution is the solution of (4.3). The graph is a students' social network [42].

For the ℓ_1-regularized PageRank problem (4.3). The Lipschitz constant used in the

proximal gradient descent (Algorithm 5) is 1 [15], while the coordinate Lipschitz constant used in the coordinate descent (Algorithm 7) is

$$\frac{1+\alpha}{2} \leq 1,$$

where $\alpha \in (0, 1]$. Therefore, if α is small, Algorithm 7 can have a larger step size than Algorithm 5.

In this research we use uniform sampling. However, with a slight change in Algorithm 7, one can handle non-uniform samplings as well. Algorithm 7 can be implemented in a parallel version, where locks are required when processing a coordinate.

Chapter 5

Accelerated Methods for ℓ_1-regularized PageRank Problem

In previous chapter, we have applied proximal gradient descent and coordinate descent to the ℓ_1-regularized PageRank problem (4.3). These methods enjoy the locality as APPR, however, still have the linear convergence rate as the proximal gradient descent method [3]. Therefore, we are interested in the performances of accelerated first order methods. To obtain an ϵ-approximate solution, the accelerated methods have convergence rate of $O(\sqrt{\frac{1}{\alpha}}\log\frac{1}{\epsilon})$ for an α-strongly convex function with Lipschitz constant 1 [3, 1].

Unfortunately, the local property of the accelerated methods introduced in this chapter has not been proved yet, because the number of nonzero vertices during the computation of accelerated methods does not have a known upper bound. Therefore, the overall iteration complexity of accelerated first order methods is undetermined.

5.1 Fast Iterative Shrinkage Algorithm (FISTA)

In Algorithm 8, FISTA has been adapted to solve the ℓ_1-regularized problem (4.3). It is implemented in the way that only nonzero vertices and their neighbors will be touched in every iteration. $\beta^{(k)}$ is a variable that controls the magnitude of the momentum in the accelerated method. In the original FISTA [4], $\beta^{(k)}$ changes in every iteration and is computed as

$$\beta^{(k)} := \frac{t^{(k)}-1}{t^{(k+1)}},$$

where $t^{(k+1)} = \frac{1+\sqrt{1+4(t^{(k)})^2}}{2}$ and $t^{(0)} = 1$. Beck and Teboulle [3] propose a modification of FISTA, where $\beta^{(k)}$ can be reset to zero when the objective start to oscillate. In the work of Chambolle et al. and Guo et al. [9, 18], it is shown that different settings of $\beta^{(k)}$ can be used to guarantee the convergence of FISTA.

Algorithm 8 Accelerated proximal gradient descent (FISTA)

procedure MINIMIZE($q, \alpha, \rho, \epsilon, s, A, D$)
  $t \leftarrow 1$;
  $q \leftarrow \vec{0}$;
  $y \leftarrow \vec{0}$;
  $\nabla f(y) \leftarrow -\alpha D^{-\frac{1}{2}} s$;
  while $\|D^{-\frac{1}{2}} \nabla f(y)\|_\infty > \rho\alpha(1+\epsilon)$ **do**
    for $i \in \{i | \|D^{-\frac{1}{2}} \nabla_i f(y)\|^{(1)} > \rho\alpha(1+\epsilon)\}$ **do**

$$\Delta q_i \leftarrow \begin{cases} y_i - \nabla_i f(y) - \rho\alpha d_i^{\frac{1}{2}} - q_i, & \text{if } y_i - \nabla_i f(y) > \rho\alpha d_i^{\frac{1}{2}}; \\ y_i - \nabla_i f(y) + \rho\alpha d_i^{\frac{1}{2}} - q_i, & \text{if } y_i - \nabla_i f(y) < -\rho\alpha d_i^{\frac{1}{2}}; \\ -q_i, & \text{otherwise.} \end{cases}$$

      $q_i \leftarrow q_i + \Delta q_i$;
      $\Delta y_i \leftarrow q_i + \beta \Delta q_i - y_i$;
      $y_i \leftarrow y_i + \Delta y_i$;
      $\nabla_i f(y) \leftarrow \nabla_i f(y) + \frac{1+\alpha}{2} \Delta y_i$;
      for vertex j such that $i \sim j$ **do**
        $\nabla_j f(y) \leftarrow \nabla_j f(y) - \frac{(1-\alpha) A_{ij}}{2 d_i^{\frac{1}{2}} d_j^{\frac{1}{2}}} \Delta y_i$;
  return $p := D^{\frac{1}{2}} q$;

5.2 Linear Coupling Method

Although many variations of the first order method have been developed, almost all such methods fundamentally rely on two types of algorithmic steps — gradient descent [8] and mirror descent[1]. Allen et al. [1] introduce an accelerated algorithm by linear coupling the two. This method has convergence rate of $O(\sqrt{\frac{1}{\alpha}} \log \frac{1}{\epsilon})$ and can be easily applied to

[1]Mirror descent has many variations [5, 29]. The linear coupling method discussed here is based on Nemirovski's mirror descent [5].

proximal gradient descent. Algorithm 9 shows the linear coupling method for solving the ℓ_1-regularized problem[2].

The key idea of the linear coupling Algorithm 9 is constructing two sequences, one for gradient descent and one for mirror descent. Thus in every single iteration k, it performs both a gradient and a mirror descent step, and ensures the two steps are coupled together at step 6. It also adopts the idea from Nesterov's accelerated method [27], where a linear combination of $y^{(k)}$ and $z^{(k)}$ is used to compute the gradient.

Algorithm 9 Linear coupling

1: **procedure** MINIMIZE($x^{(0)}, \rho, \alpha, D$)
2: $y^{(0)} \leftarrow x^{(0)}$;
3: $z^{(0)} \leftarrow x^{(0)}$;
4: **for** $k = 0, 1, \ldots, K-1$ **do**
5: $\gamma^{(k)} \leftarrow \frac{k+2}{2}$;
6: $\tau^{(k)} \leftarrow \frac{1}{\gamma^{(k)}}$;
7: $x^{(k+1)} \leftarrow \tau^{(k)} z^{(k)} + (1-\tau^{(k)}) y^{(k)}$;
8: $y^{(k+1)} \leftarrow \operatorname{argmin}_y \left\{ \rho\alpha \|D^{\frac{1}{2}} y\|_1 + \nabla f(x^{(k+1)})^T (y - x^{(k+1)}) + \frac{1}{2}\|y - x^{(k+1)}\|_2^2 \right\}$;
9: $z^{(k+1)} \leftarrow \operatorname{argmin}_z \left\{ \rho\alpha \|D^{\frac{1}{2}} z\|_1 + \nabla f(x^{(k+1)})^T (z - z^{(k)}) + \frac{1}{2\gamma^{(k)}}\|z - z^{(k)}\|_2^2 \right\}$;
10: **return** $x^{(K)}$;

5.3 Unknown Locality of Accelerated Methods

In this section, we discuss the open question: how to prove FISTA is a local method[3]. The main challenge is that, in FISTA, the partial derivative $\nabla_i f(q^{(k)})$ of a nonzero vertex i is no longer upper bounded by $-\rho\alpha d_i^{\frac{1}{2}}$. Hence theoretical analysis in the work of Fountoulakis et al. [15] cannot be used to bound the volume of $S^{(k)}$ defined in (4.11). As a result, there is no theoretical proof that shows FISTA is a local method, whose iteration complexity depends on the support of optimal solution instead of the size of the graph.

Recall that we still have the assumption (4.7) to ensure that there is significant probability mass in seed vertices. For simplicity, let i be the only seed vertex. In Algorithm 8,

[2]Similar to Algorithm 8, Algorithm 9 can be implemented in the way that only nonzero vertices and their neighbors will be touched in every iteration.

[3]The linear coupling method has the same question and can be explained in a similar way.

we have $q^{(0)} = \vec{0}$, $y^{(0)} = \vec{0}$ and $\nabla_i f(y^{(0)}) \leq -\rho\alpha d_i^{\frac{1}{2}}$. In the first iteration,

$$\Delta q^{(0)}(j) = \begin{cases} y^{(0)}(j) - (\nabla_j f(y^{(0)}) + \rho\alpha d_j^{\frac{1}{2}}), & \text{if } j = i \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\Delta q_i^{(0)} = -(\nabla_i f(y^{(0)}) + \rho\alpha d_i^{\frac{1}{2}}) \geq 0$. Consequently,

$$\begin{aligned} \Delta y_i^{(0)} &= (1 + \beta^{(0)})\Delta q_i^{(0)} \\ &> 0. \end{aligned}$$

The gradient in the next iteration, denoted as $\nabla_i f(y^{(1)})$, is updated as

$$\begin{aligned} \nabla_i f(y^{(1)}) &= \nabla_i f(y^{(0)}) + \frac{1+\alpha}{2}\Delta y_i^{(0)} \\ &= \nabla_i f(y^{(0)}) + \frac{(1+\alpha)(1+\beta^{(0)})}{2}\Delta q_i^{(0)}, \end{aligned}$$

Since $\beta^{(k)} = \frac{t^{(k)}-1}{t^{(k+1)}}$ and $t^{(0)} = 1$, we have

$$\begin{aligned} \nabla_i f(y^{(1)}) &= \nabla_i f(y^{(0)}) + \frac{1+\alpha}{2}\Delta q_i^{(0)} \\ &= \nabla_i f(y^{(0)}) + \Delta q_i^{(0)} - \frac{1-\alpha}{2}\Delta q_i^{(0)} \\ &= -\rho\alpha d_i^{\frac{1}{2}} - \frac{1-\alpha}{2}\Delta q_i^{(0)} \\ &< -\rho\alpha d_i^{\frac{1}{2}}. \end{aligned}$$

Until now, we have $q_i^{(1)} \geq 0$, $y_i^{(1)} \geq 0$ and $\nabla_i f(y^{(1)}) \leq -\rho\alpha d_i^{\frac{1}{2}}$. Then at iteration $k = 1$, we get

$$\begin{aligned} \Delta q_i^{(1)} &= -(\nabla_i f(y^{(1)}) + \rho\alpha d_i^{\frac{1}{2}}) \\ &> 0. \end{aligned}$$

Thus the Δy_i is computed as

$$\Delta y_i^{(1)} = q_i^{(1)} + \beta\Delta q_i^{(1)} - y_i^{(1)},$$

which is not necessarily positive. As a result, the sign of $\nabla_i f(y^{(2)}) = \nabla_i f(y^{(1)}) + \frac{1+\alpha}{2}\Delta y_i^{(1)}$ can not be determined. Figure 5.1 and 5.2 illustrate that accelerated methods converge

fast, but their number of nonzero vertices exceeds the number of nonzero vertices in the optimal solution.

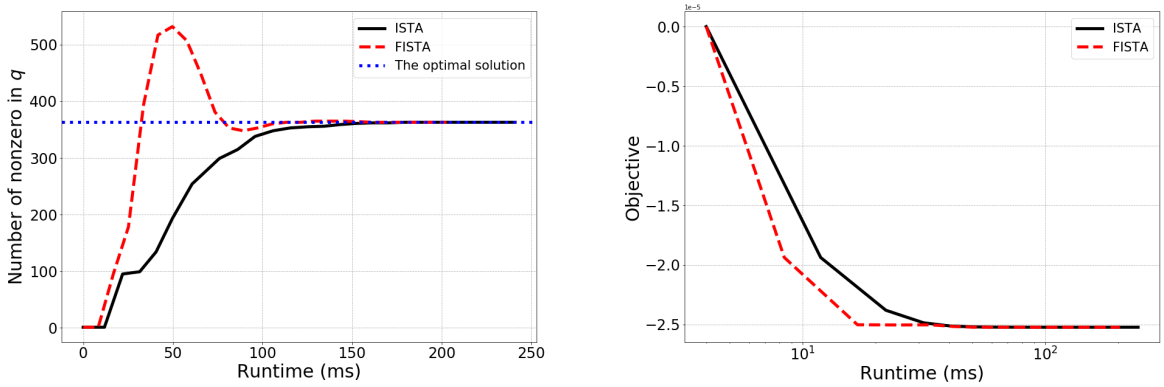

Figure 5.1: This figure shows the performances of FISTA and ISTA in solving the ℓ_1-regularized problem (4.3). The optimal solution is the solution of (4.3). The graph is a students' social network [42].

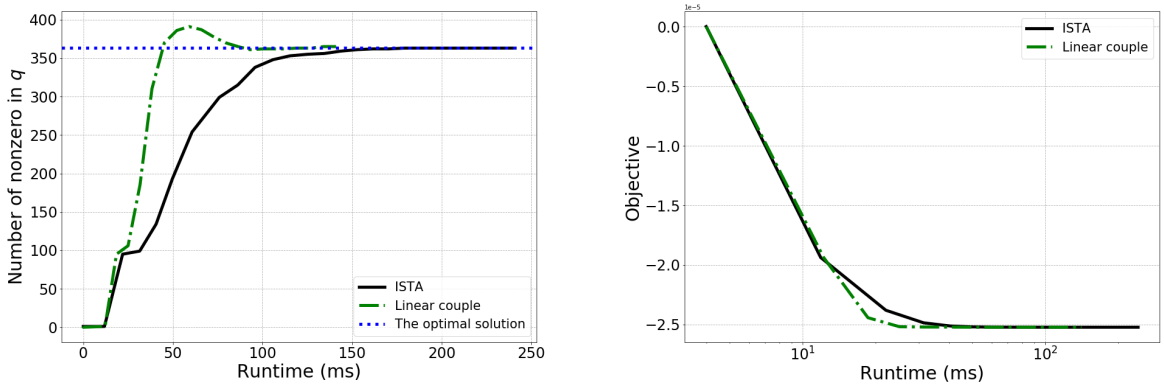

Figure 5.2: This figure shows the performances of the linear coupling method and ISTA in solving the ℓ_1-regularized problem (4.3). The optimal solution is the solution of (4.3). The graph is a students' social network [42].

Chapter 6

Computational Results

In this chapter, we present the numerical performance of algorithms in large-scale graphs. We also implement two accelerated methods that can be applied to solve the ℓ_1-regularized PageRank problem. The experiments are performed on a Linux machine with twelve 2.6GHz Intel i7-6770HQ processors and 120GB RAM. All algorithms are implemented in C++ and compiled with the g++ compiler of version 4.8.4.

Table 6.1 shows the metrics of the graphs. Code to reproduce all experiments can be found at the GitLab repository. Following is a list of short descriptions of these graphs

- **JohnHopkins**. Facebook anonymized data set on a particular day in September 2005 for a student social network at John Hopkins University. [42].
- **Senate**. Each node in this network is a Senator that served in a single term (two years) of Congress. The data cover the period from year 1789 to 2008 [26].
- **usroads**. A graph from the National Highway Planning Network. Each vertex in this network is an intersection between two highways and the edges represent segments of the highways themselves [6].
- **com-Youtube**. Youtube is a video-sharing web site that includes a social network. In the Youtube social network, users form friendship each other and users can create groups which other users can join [45].
- **cit-Patents**. U.S. patent data set is maintained by the National Bureau of Economic Research. The data set spans 37 years, and includes all the utility patents granted during that period [22].

- **com-LiveJournal**. LiveJournal is a free online blogging community where users can declare friendship each other [45].
- **com-Orkut**. Orkut is a free online social network where users form friendship each other. [45].

Graph name	Num. vertices	Num. edges
JohnHopkins	5,157	186,585
Senate	8,974	153,804
usroads	126,145	323,899
com-Youtube	1,134,890	2,987,624
cit-Patents	3,774,768	16,518,948
com-LiveJournal	3,997,962	34,681,189
com-Orkut	3,072,441	117,185,083

Table 6.1: The graphs used in this experiment

6.1 Experiment Settings

We measure the performance of ISTA, random coordinate descent (RCD), FISTA, linear coupling method (LC), FISTA with constant step size (FISTA CONST), and linear coupling with restart technique (LC RESTART).

Since the function F defined in (4.3) is α-strongly convex [15]. Therefore, the $\beta^{(k)}$ in FISTA can be simplified to a constant as follows [3]

$$\beta^{(k)} = \frac{1-\sqrt{\alpha}}{1+\sqrt{\alpha}}.$$

The linear coupling method with the restart technique is implemented such that the iteration count k will be reset to 0 when $k \geq \sqrt{\frac{1}{\alpha}}$. According to the analysis derived by Allen et al. and O'donoghue et al.[1, 33], such heuristic restart technique has better theoretical convergence rate than the original method.

Throughout this experiment, we always use a single vertex as the seed vertex (i.e. we set $s(v) = 1$ and 0 elsewhere, where v is the seed vertex). We measure the average running time of algorithms starting with different seed vertices. For small graphs (i.e.

JohnHopkins, usroads), every vertex in the graph is tested as the seed vertex. For large graphs (i.e. com-Youtube, cit-Patents, com-LiveJournal, com-Orkut), we sample over 10^5 vertices in the graphs. With the sweep method, we make sure that all seed vertices return clusters with at least 5 vertices.

When measuring the performance of algorithms with different α, we set $\rho = 10^{-5}$. When measuring the performance of algorithms with different ρ, we set $\alpha = 0.05$. We also measure how the running time scales as the number of nonzero vertices in the optimal solution increases. All presented plots show the averaged running time and the standard deviation. To illustrate the locality of algorithms, we plot the number of nonzero vertices against the iteration counter k, and the optimal solution is the solution of (4.3). The seed vertex is randomly picked from the graphs.

6.2 Computational Results

6.2.1 Number of Nonzero Vertices

In the first experiment we investigate, with a random seed vertex, the number of nonzero vertices visited by 6 algorithms. Looking at Table 6.2, we see that the number of nonzero vertices of ISTA and random implies descent is much less than the size of the graphs, which indicates the algorithms are local. This is because it has been shown that ISTA and random coordinate descent touch vertices that will be included in the support of the optimal solution [15]. Although FISTA, linear coupling, and their variants may touch vertices that are not in the optimal solution, these algorithms only visit a small part of the graphs.

Methods	usroads	com-Youtube	cit-Patents	com-LJ	com-Orkut
ISTA	282	202	2,297	3,320	1,247
RCD	282	202	2,297	3,320	1,247
FISTA	316	214	2,304	3,778	1,356
LC	301	210	2,313	3,436	1,391
FISTA CONST	282	214	2,298	3,986	1,391
LC RESTART	282	226	2,313	3,436	1,247

Table 6.2: This table shows the number of nonzero vertices encountered using 6 algorithms. $\alpha = 0.05$, $\rho = 10^{-5}$.

6.2.2 Regularization Parameter

Figures 6.1 and 6.2 present the results of the local graph clustering with different regularization parameters. One can observe that the sizes of output clusters depend on the magnitude of the regularization parameter ρ. As ρ decreases from 10^{-4} to 10^{-7}, the average size of returned clusters in the graph of USA road system increases from 10 to 10^3. This is reasonable since Fountoulakis et al. [15] have proved that the volume of $\mathcal{F}^*$ is upper bounded by $\frac{\|s\|_1}{\rho}$, where $\mathcal{F}^*$ is the support of the optimal solution for (4.3).

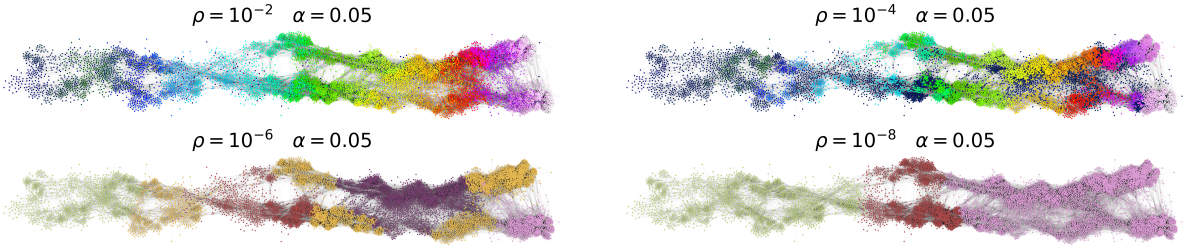

Figure 6.1: This figure shows the result of the local graph clustering in graph **Senate** [26]. Sweep method find all clusters that have at least 10 vertices from the optimal solution of (4.3). When $\rho = 10^{-8}$, there are 3 large clusters each with thousands of vertices.

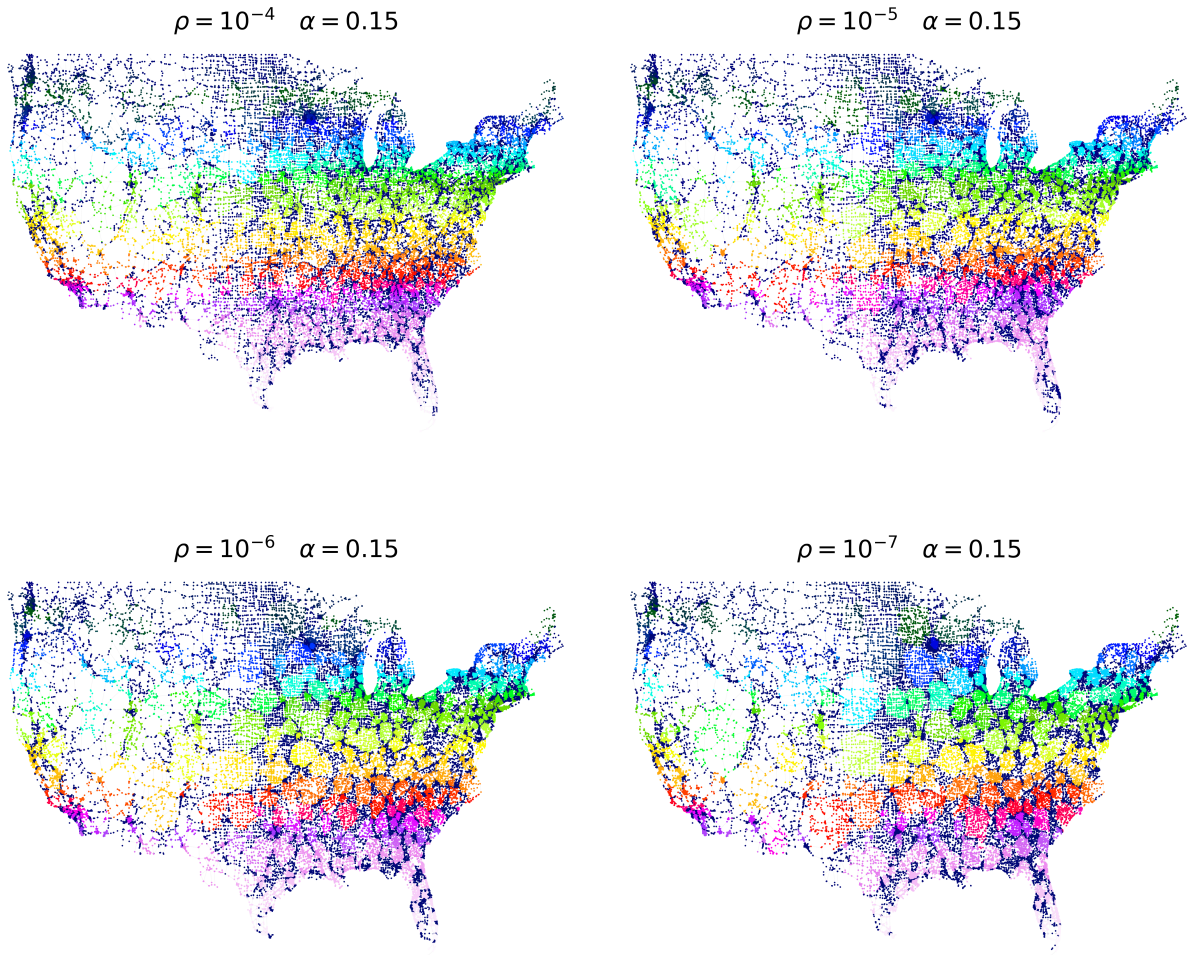

Figure 6.2: This figure shows the result of the local graph clustering in the graph **usroads** [6]. Sweep method finds all clusters that have at least 10 vertices from the optimal solution of (4.3).

6.2.3 Teleport Probability

In the PageRank problem, the random surfer stays at the starting web page with probability α. Therefore, the surfer with large teleport probability is less likely to visit vertices far away from the starting vertices.

Figures 6.3 and 6.4 show that the teleport probability α affects the scale of clustering. We observe that, with a larger α, the far-flung vertices become more isolated, making the cluster denser. Similar to the clustering results of non-regularized PageRank optimization [17], a larger α leads to clusters concentrated near the seed vertices.

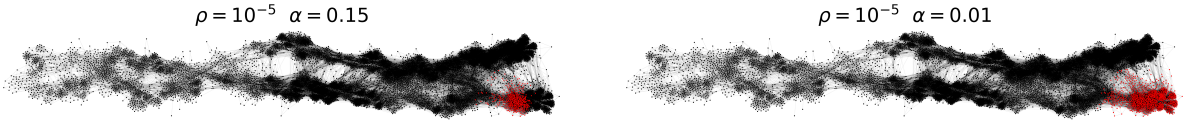

Figure 6.3: This figure shows the local clustering result in the graph **Senate** [26]. The cluster is found by the sweep method applied to the optimal solution of (4.3).

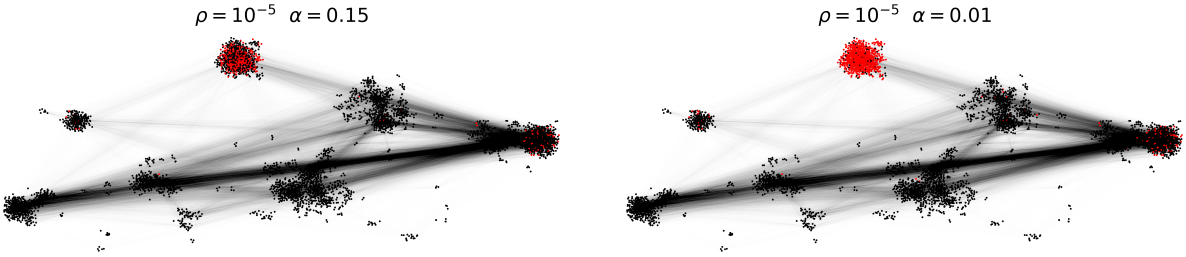

Figure 6.4: This figure shows the local graph clustering in the graph of **JohnHopkins** [42]. The cluster is found by the sweep method applied to the optimal solution of (4.3).

6.2.4 Running Time

Since we never observe accelerated methods touch a surprisingly large number of nonzero vertices in the experiment, we expect that accelerated methods are more efficient than non-accelerated methods in solving (4.3).

Let us first look at the smaller of the graphs: **JohnHopkins**, **usroads** and **Senate** (Figure 6.5, 6.6 and 6.7). When $\rho \leq 5 \times 10^{-6}$ and $\alpha \leq 0.2$, we notice that FISTA CONST significantly outperforms the other methods. When $\rho > 10^{-5}$ and $\alpha > 0.6$, algorithms converge in a few milliseconds and have similar performance. In the figure of nonzero vertices against iteration counter k (i.e. bottom right subplots of Figure 6.5 and 6.6), we observe that algorithms use almost the same number of iterations to converge. However

the time to initialize and manipulate additional sequences in accelerated methods becomes more significant when the problem is easy to solve.

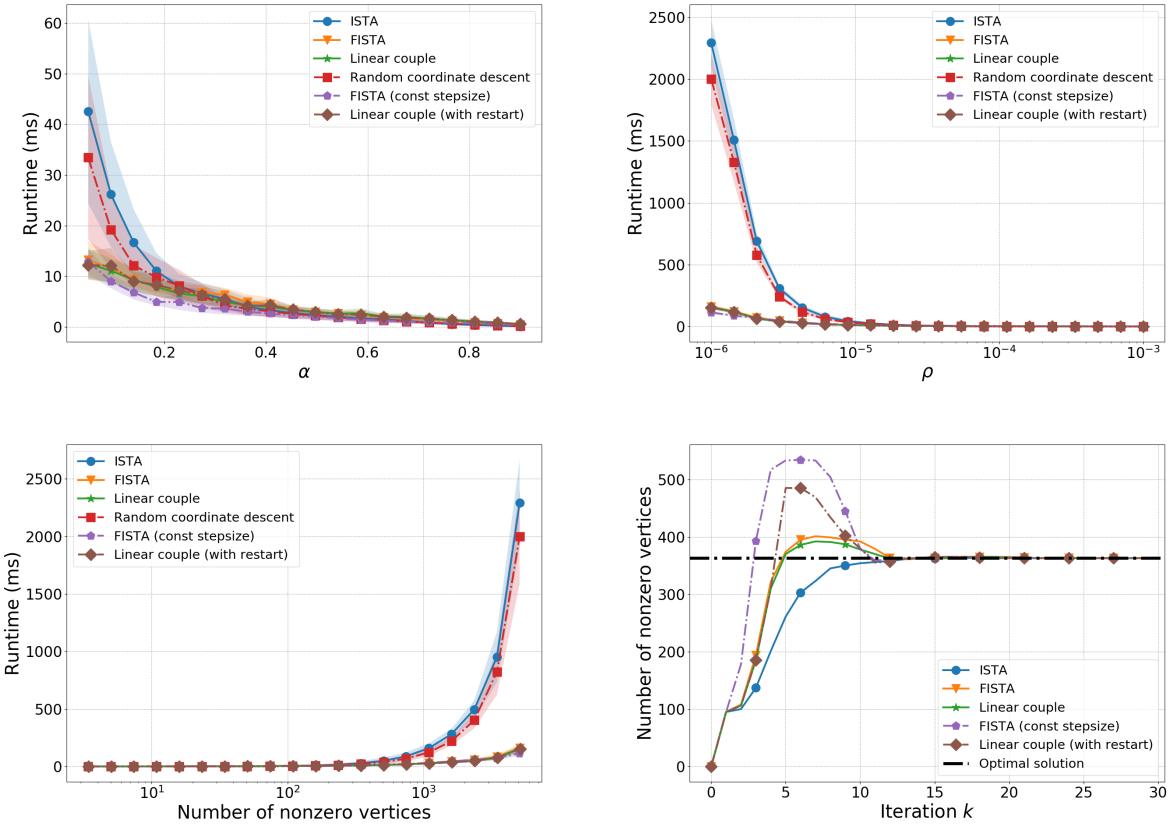

Figure 6.5: These plots show the performance of 6 algorithms in the graph **JohnsHopkins**[42].

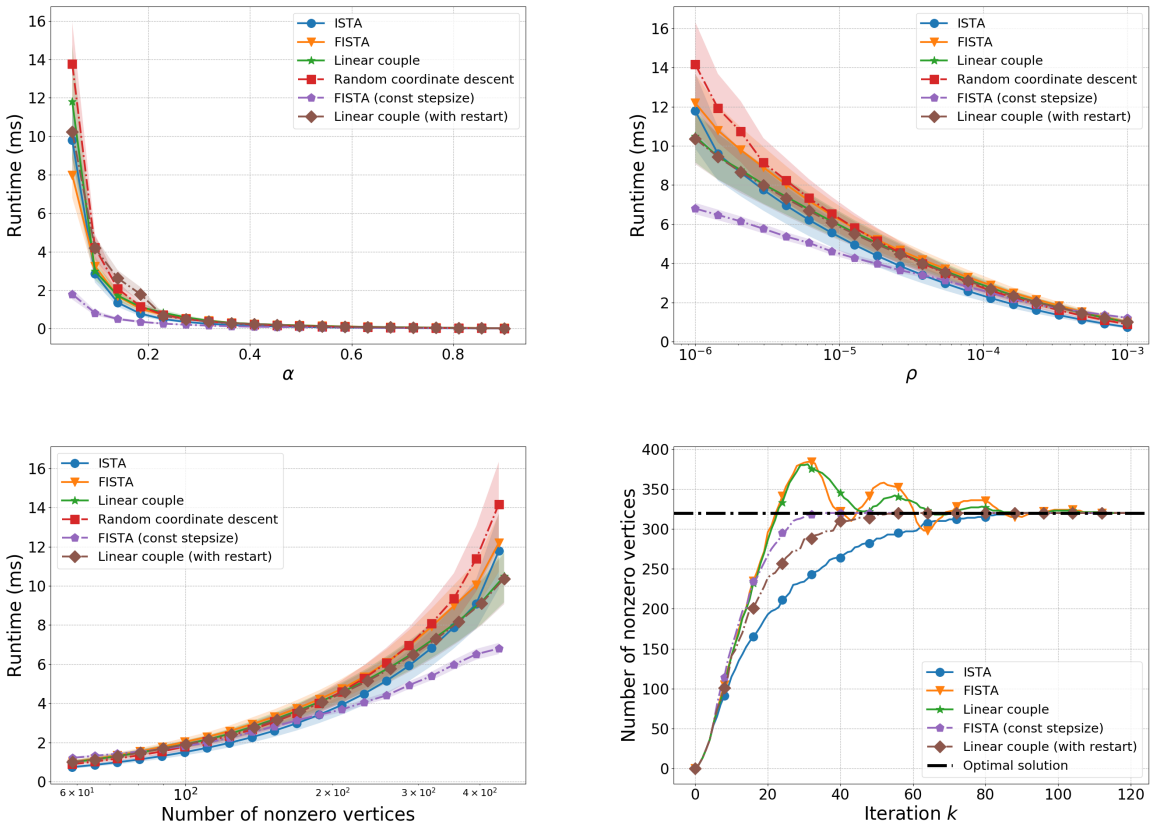

Figure 6.6: These plots show the performance of 6 algorithms in the graph **usroads** [6].

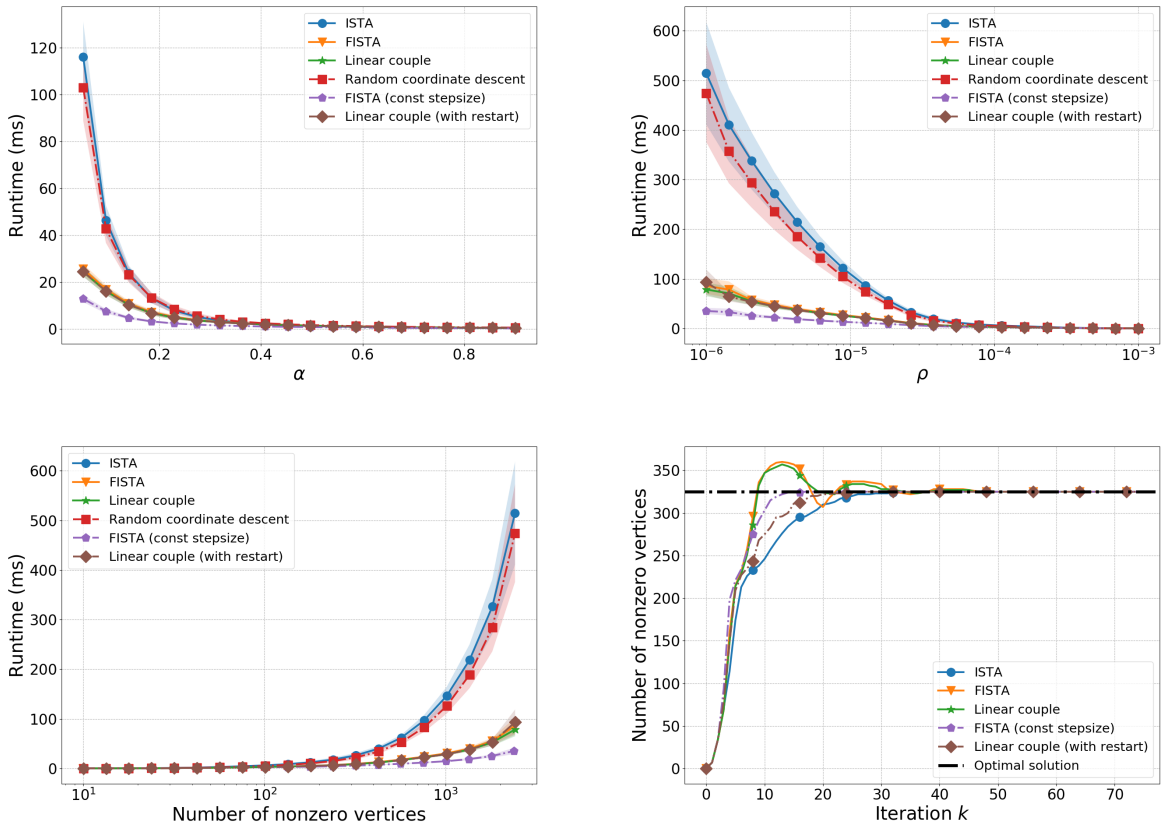

Figure 6.7: These plots show the performance of 6 algorithms in the graph **Senate** [26].

By looking at the results of the large graphs: **com-Youtube**, **cit-Patents** and **com-LJ** (Figure 6.8, 6.9 and 6.10), we find that RCD often outperforms ISTA. When the number of nonzero vertices in the optimal solution is over 10^3 vertices, the running time of accelerated methods are consistently less than the running time of non-accelerated methods. When the number of nonzero vertices exceeds 4×10^4, the differences among performance of algorithms become significant. Looking at Fig 6.10, with $\rho = 10^{-6}$ and $\alpha = 0.05$, the average time to converge to the optimal solution over 399,796 vertices for ISTA, FISTA, and linear coupling method are, respectively, 2.82024 sec, 1.08339 sec, and 0.89591 sec. In the graph **com-Youtube**, when $\rho \approx 5 \times 10^{-6}$, the running time of ISTA is still worse than the running time of accelerated methods with $\rho = 10^{-6}$. In the graph **cit-Patents**, FISTA CONST uses 0.65164 seconds to converge to an optimal solution with 12,756 nonzero vertices, while ISTA uses 0.96864 seconds to converge to an optimal solution with 6,630

nonzero vertices.

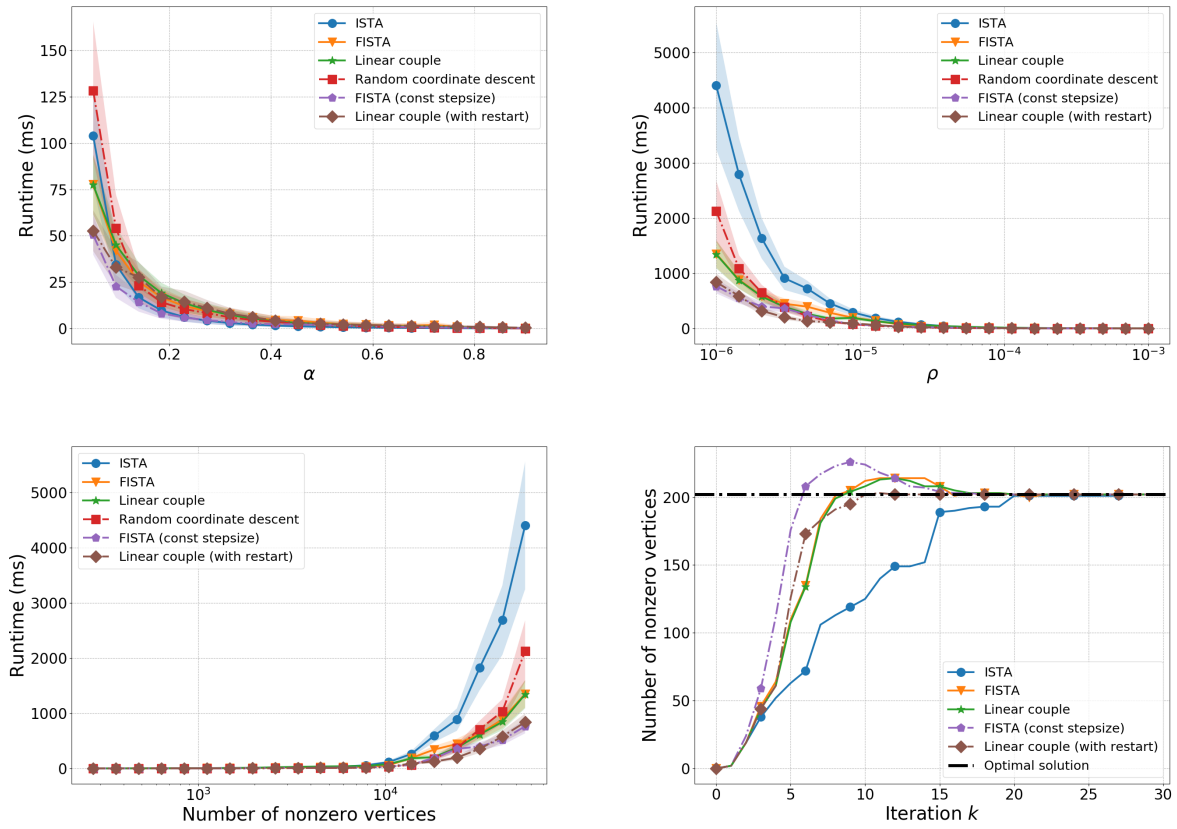

Figure 6.8: These plots show the performance of 6 algorithms in the graph **com-Youtube** [45].

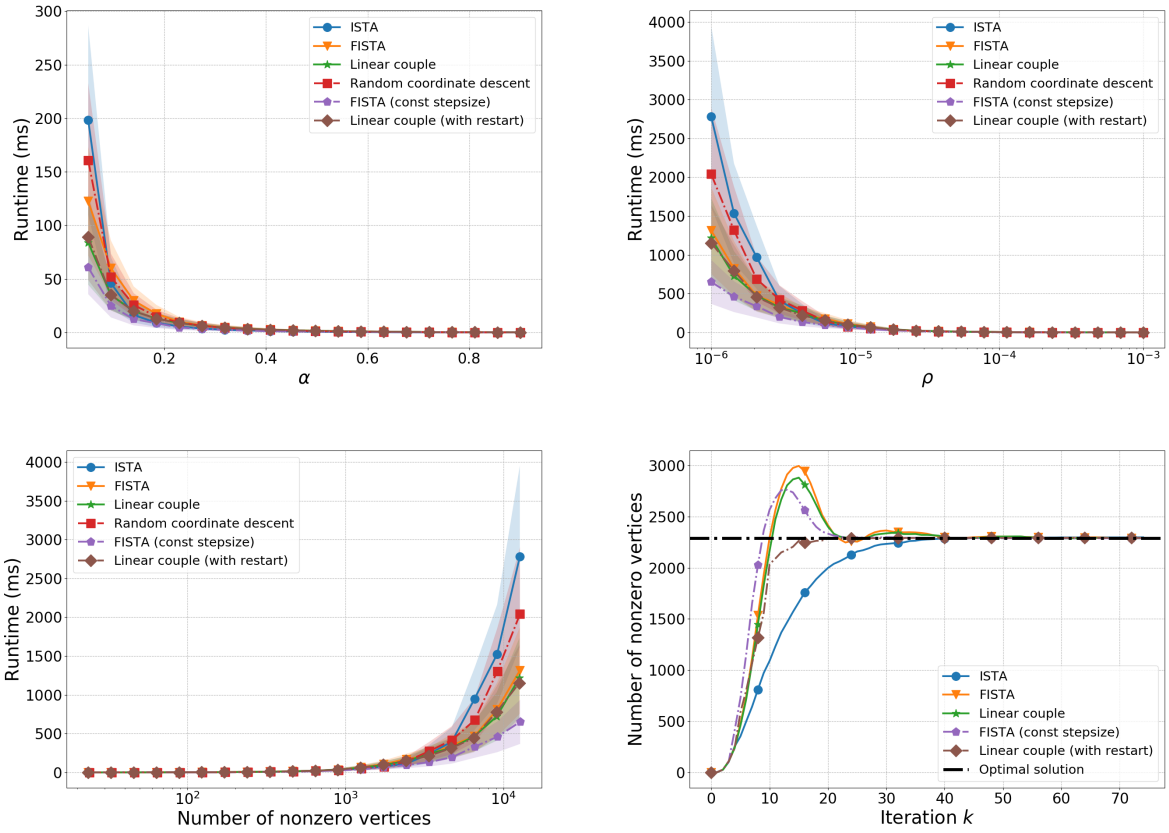

Figure 6.9: These plots show the performance of 6 algorithms in the graph **cit-Patents** [22].

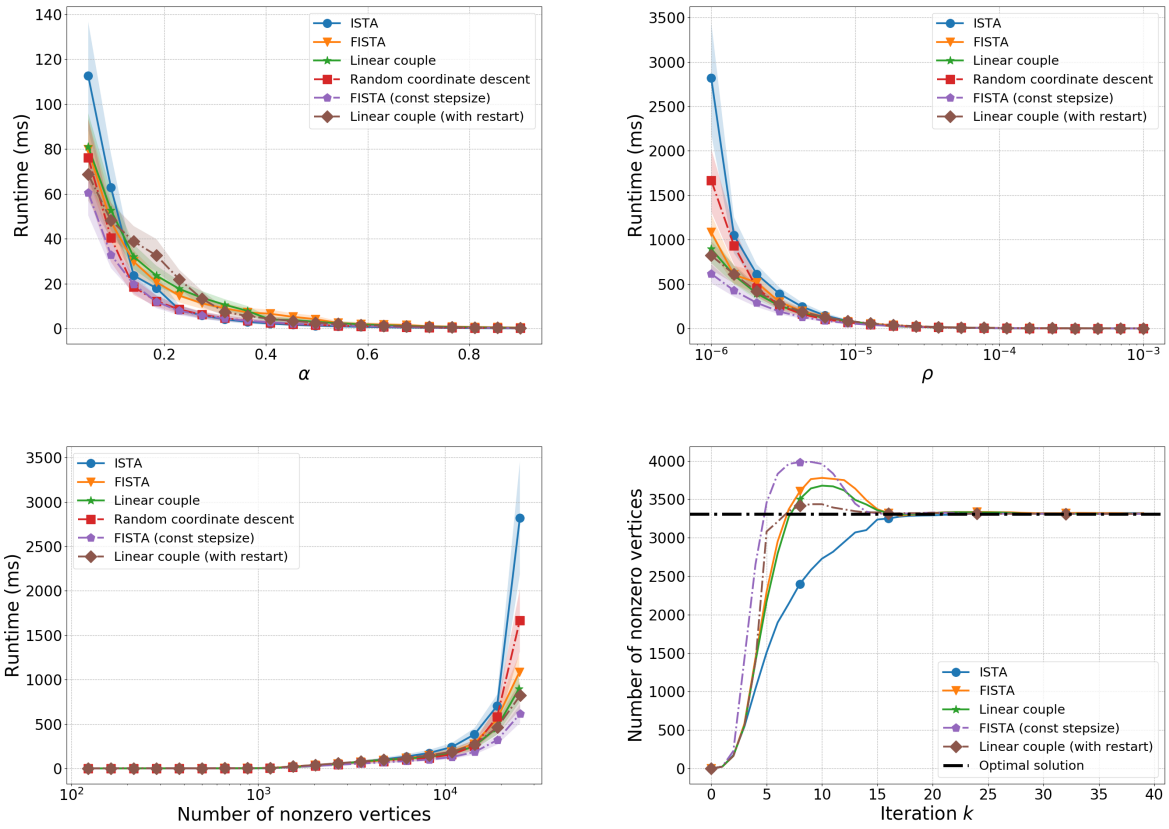

Figure 6.10: These plots show the performance of 6 algorithms in the graph **com-LiveJournal** [45].

In Figure 6.11, we notice that **com-Orkut** is the only large graph where non-accelerated methods can be more efficient than accelerated methods. Compared with other graphs in this experiment, **com-Orkut** has the most communities (8,455,253) and its vertices have larger degrees on average. It is interesting to see that RCD outperforms accelerated methods when the number of nonzero vertices in the optimal solution is less than 1.2×10^4. One reason of this behavior is that accelerated methods are likely to process much more vertices than non-accelerated in a dense graph like **com-Orkut**. However, when the solution becomes less sparse, it spends a lot of time in finding the best coordinate to update.

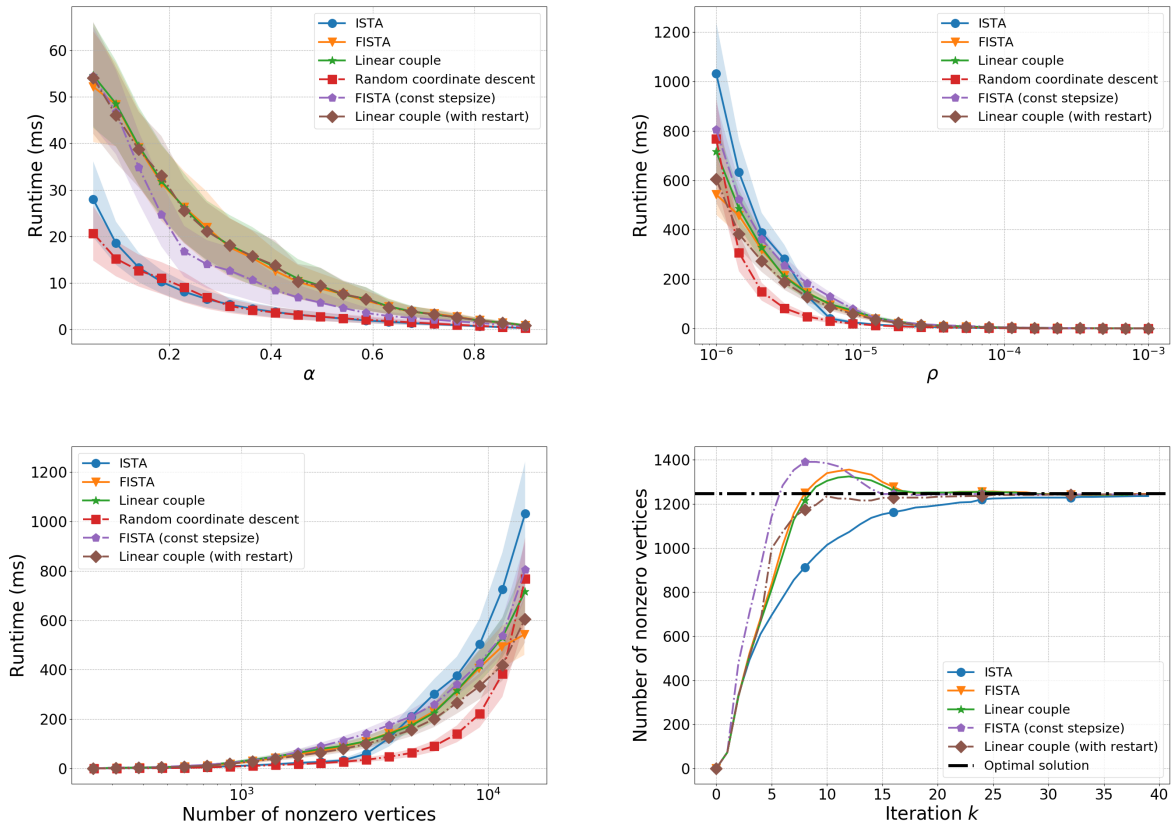

Figure 6.11: These plots show the performance of 6 algorithms in the graph **com-Orkut** [45].

Now we will further show the benefit of accelerated methods in solving the ℓ_1-regularized PageRank problem. In some applications, rather than finding a single cluster, one is interested in middle- and small-scale structures of large graphs. Therefore, in Table 6.3, we demonstrate the running time of 6 algorithms for finding over 10^3 clusters in **com-Orkut**, **com-Youtube** and **com-LJ**. The algorithms are tested with regularization parameter $\rho = 10^{-6}$, teleport probability $\alpha = 0.05$, and 2×10^4 starting vertices. Clearly, accelerated methods converge faster than non-accelerated methods. Moreover, in the graph **com-Youtube**, ISTA needs 679 minutes to find clusters that FISTA CONST obtained after 175 minutes.

Methods	com-Orkut (3,630 clusters)	com-Youtube (4,803 clusters)	com-LJ (5,269 clusters)
ISTA	510.44	679.04	1,014.44
RCD	438.20	583.20	792.94
FISTA	436.76	316.07	488.87
LC	366.22	317.20	534.32
FISTA CONST	342.38	**175.41**	**319.32**
LC RESTART	**280.90**	299.69	481.27

Table 6.3: This table shows the running time (min) of 6 algorithms for finding clusters in **com-Orkut**, **com-Youtube** and **com-LJ**.

Chapter 7

Conclusion

In this thesis, we study the randomized coordinate descent method when solving the ℓ_1-regularized PageRank problem. We prove that the number of nonzero vertices in the coordinate decent method depends on the support of the optimal solution. We derive the convergence of the coordinate descent and the computational result shows that it outperforms the proximal gradient descent proposed in the work of Fountoulakis et al. [15] for solving the same problem.

Although showing an upper bound for the number of nonzero vertices in the accelerated methods remains to be an interesting challenge, the accelerated methods and their variants have the best performance when the support of optimal solution is sufficiently large. Moreover, the computational results show that FISTA CONST and LC RESTART are more efficient than their original versions in the three large graphs.

The comparison between algorithms demonstrates that:

- In large-scale graphs, FISTA CONST has the better performance than other 5 algorithms measured in our experiment.
- When $\rho \approx 10^{-6}$ and $\alpha \approx 0.05$, accelerated methods converge faster than non-accelerated method, and ISTA is the most expensive approach.
- RCD outperforms ISTA in large graphs. Additionally, when the vertices have larger degrees on average, RCD may converge faster than accelerated methods.
- Although the set of nonzero vertices of accelerated methods can exceed the support of optimal solution q^*. Accelerated methods have better performances when $|supp(q^*)| \approx 10^4$.

Future efforts will be made in the following aspects. More complicated sampling strategies in the random coordinate descent can be explored. For example, Fountoulakis et al. [15] mention a heuristic way of selecting the coordinate with largest $\nabla_i f(q^{(k)})$ in absolute value. Another possible approach is sampling coordinate i with probability $\frac{d_i}{\sum_{j\in[m]} d_j}$, because vertices with more edges are more likely to assign probability to their neighbors. Also, advanced analysis can be used to find the upper bound for the number of nonzero vertices visited by FISTA or linear coupling method. Alternatively, one may consider the local properties in Theorem 1 as constraints, and apply algorithms that solve the constrained optimization problem efficiently.

References

[1] Zeyuan Allen-Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. *arXiv preprint arXiv:1407.1537*, 2014.

[2] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[3] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[4] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

[5] Charles Blair. Problem complexity and method efficiency in optimization (as nemirovsky and db yudin). *SIAM Review*, 27(2):264, 1985.

[6] Geoff Boeing. U.S. Street Network Shapefiles, Node/Edge Lists, and GraphML Files, 2017.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[8] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[9] Antonin Chambolle and Ch Dossal. On the convergence of the iterates of the "fast iterative shrinkage/thresholding algorithm". *Journal of Optimization theory and Applications*, 166(3):968–982, 2015.

[10] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. American Mathematical Soc., 1997.

[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 991–1002, 2014.

[12] John C Duchi, Shai Shalev-Shwartz, Yoram Singer, and Ambuj Tewari. Composite objective mirror descent. In *COLT*, pages 14–26, 2010.

[13] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[14] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern recognition*, 41(1):176–190, 2008.

[15] Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W Mahoney. Variational perspective on local graph clustering. *Mathematical Programming*, 174(1-2):553–573, 2019.

[16] David F Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015.

[17] Fan Chung Graham and Alexander Tsiatas. Finding and visualizing graph clusters using pagerank optimization. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 86–97. Springer, 2010.

[18] Ke Guo, Xiaoming Yuan, and Shangzhi Zeng. Convergence analysis of ista and fista for "strongly+ semi" convex programming, 2016.

[19] Wooseok Ha, Kimon Fountoulakis, and Michael W Mahoney. Statistical guarantees for local graph clustering. *arXiv preprint arXiv:1906.04863*, 2019.

[20] John Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 541–546. ACM, 2003.

[21] Kevin Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 325–337. Springer, 2004.

[22] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[23] Michael W Mahoney, Lorenzo Orecchia, and Nisheeth K Vishnoi. A spectral algorithm for improving graph partitions. Technical report, Technical report. Preprint, 2009.

[24] Michael W Mahoney, Lorenzo Orecchia, and Nisheeth K Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *Journal of Machine Learning Research*, 13(Aug):2339–2365, 2012.

[25] Hans-Werner Mewes, Dmitrij Frishman, Ulrich Güldener, Gertrud Mannhaupt, Klaus Mayer, Martin Mokrejs, Burkhard Morgenstern, Martin Münsterkötter, Stephen Rudd, and B Weil. Mips: a database for genomes and protein sequences. *Nucleic acids research*, 30(1):31–34, 2002.

[26] Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328(5980):876–878, 2010.

[27] Y Nesterov. A method of solving a convex programming problem with convergence rate $O(\frac{1}{k^2})$. In *Soviet Math. Dokl*, volume 27, 1983.

[28] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[29] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[30] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[31] Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[32] Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1267–1286. SIAM, 2014.

[33] Brendan O'donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

[34] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[35] R Tyrrell Rockafellar. *Convex analysis*. Number 28. Princeton university press, 1970.

[36] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[37] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W Mahoney. Parallel local graph clustering. *Proceedings of the VLDB Endowment*, 9(12):1041–1052, 2016.

[38] Daniel A Spielman. Spectral graph theory and its applications. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 29–38. IEEE, 2007.

[39] Daniel A Spielman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 96–105. IEEE, 1996.

[40] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013.

[41] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518, 2014.

[42] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

[43] Nate Veldt, David Gleich, and Michael Mahoney. A simple and strongly-local flow-based method for cut improvement. In *International Conference on Machine Learning*, pages 1938–1947, 2016.

[44] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[45] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth, 2012.

APPENDICES

Appendix A

Python implementations

A.1 Approximate Personalized PageRank (APPR)

Below is the Python implementation of algorithm 4.

```
def add_candidate(node, r, epsilon, candidate_nodes, graph):
    if r[node] >= epsilon*graph.d[node]:
        candidate_nodes.add(node)


def push(node, alpha, epsilon, candidate_nodes, p, r, A, graph):
    p[node] += alpha*r[node]
    for neighbor in graph.neighbors(node):
        r[neighbor] += \
            .5*(1-alpha)*r[node]*A[node,neighbor]/graph.d[node]

        add_candidate(neighbor,r,epsilon,candidate_nodes,graph)
    r[node] *= .5*(1-alpha)
    add_candidate(node,r,epsilon,candidate_nodes,graph)


def appr(alpha, epsilon, seed_node, graph):
    A = graph.adjacency_matrix.tocsc()
    candidate_nodes = set([seed_node])
    p = np.zeros(graph._num_vertices)
```

```
    r = np.zeros(graph._num_vertices)
    r[seed_node] = 1
    while candidate_nodes:
        node = candidate_nodes.pop()
        push(node,alpha,epsilon,candidate_nodes,p,r,A,graph)
    return p
```

A.2 Sweep clustering

Below is the Python implementation of sweep technique.

```
def sweep_cut(p, graph):
    cluster = set()
    min_cut, min_cond, cnt_leave, cnt_degree = len(p), 1, 0, 0
    vec = sorted(\
        [(p[i]/graph.d[i], i) for i in range(len(p))],\
         reverse=True)

    for i, (val, node) in enumerate(vec):
        if val == 0: break
        cluster.add(node)
        cnt_degree += graph.d[node]
        for neighbor in graph.neighbors(node):
            cnt_leave += 1 if neighbor not in cluster else -1
        if cnt_leave/cnt_degree < min_cond:
            min_cond, min_cut = cnt_leave/cnt_degree, i

    return [vec[i][1] for i in range(min_cut+1)]
```

A.3 ISTA

Below is the Python implementation of ISTA for solving the ℓ_1-regularized PageRank problem

```
def get_gradient(alpha, q, dn_sqrt, s, Q):
    return Q@q - alpha*dn_sqrt*s

def proximal_step(alpha, rho, q, gradient, d_sqrt):
    x = np.zeros(len(q))
    threshold = rho*alpha*d_sqrt
    for i in range(len(q)):
        if q[i] - gradient[i] >= threshold[i]:
            x[i] = q[i] - gradient[i] - threshold[i]
        elif q[i] - gradient[i] <= -threshold[i]:
            x[i] = q[i] - gradient[i] + threshold[i]
        else:
            x[i] = 0
    return x

def ISTA(alpha, rho, d_sqrt, dn_sqrt, s, Q, max_iter = 20):
    q = np.zeros(len(dn_sqrt))

    for _ in range(max_iter):
        gradient = get_gradient(alpha, q, dn_sqrt, s, Q)
        q = proximal_step(alpha, rho, q, gradient, d_sqrt)

    return q
```

A.4 FISTA

Below is the Python implementation of FISTA for solving the ℓ_1-regularized PageRank problem

```
def FISTA(alpha, rho, d_sqrt, dn_sqrt, s, Q, max_iter = 20):
    q = np.zeros(len(dn_sqrt))
    y = np.zeros(len(dn_sqrt))
    t = 1

    for num_iter in range(max_iter):
        grad = get_gradient(alpha, y, dn_sqrt, s, Q)
```

```
        prev_q, q = q, proximal_step(alpha,rho,y,grad,d_sqrt)
        prev_t, t = t, (1+np.sqrt(1+4*t**2))/2
        y = q+(1-np.sqrt(alpha))/(1+np.sqrt(alpha))*(q-prev_q)

    return q
```

A.5 Linear Coupling Method

Below is the Python implementation of linear coupling method for solving the ℓ_1-regularized PageRank problem

```
def update_yz(alpha, rho, q, gradient, d_sqrt):
    yz = np.zeros(len(q))
    threshold = rho*alpha*d_sqrt
    for i in range(len(q)):
        if q[i] - gradient[i] >= threshold[i]:
            yz[i] = q[i] - gradient[i] - threshold[i]
    return yz

def MirrorDescent(alpha, rho, d_sqrt, dn_sqrt, s, Q, max_iter):
    q = np.zeros(len(dn_sqrt))
    z = np.zeros(len(dn_sqrt))
    y = np.zeros(len(dn_sqrt))

    for _ in range(max_iter):
        gamma = (num_iter+2)/2.0
        tau = 1/gamma
        q = tau*z + (1-tau)*y
        grad = get_gradient(alpha, q, dn_sqrt, s, Q)
        y = update_yz(alpha, rho, q, grad, d_sqrt)
        z = update_yz(gamma*alpha, rho, z, gamma*grad, d_sqrt)
        tau = 2.0/(num_iter+3)

    return q
```

A.6 Random Coordinate Descent

Below is the Python implementation of random coordinate descent for solving the ℓ_1-regularized PageRank problem

```
import random

def update(i, q, gradient, nonzero_set, alpha,\
           rho, Dnsqrt, Dsqrt, s, graph):
    dq = -gradient[i] - rho*alpha*Dsqrt[i]
    q[i] += dq
    gradient[i] = -rho*alpha*Dsqrt[i]-.5*(1-alpha)*dq
    for j in graph.neighbors(i):
        gradient[j] -= .5*(1-alpha)*Dnsqrt[j]*Dnsqrt[i]*dq
        if q[j] - gradient[j] >= rho*alpha*Dsqrt[j]:
            nonzero_set.add(j)

def rand_coord_descent(alpha, rho, Dnsqrt, Dsqrt,\
                       s, graph, max_iter = 100):
    q = np.zeros(len(s))
    gradient = -alpha*Dnsqrt*s
    nonzero_set = set(np.nonzero(s)[0])
    for _ in range(max_iter):
        i = random.choice(tuple(nonzero_set))
        update(i, q, gradient, nonzero_set,\
               alpha, rho, Dnsqrt, Dsqrt, s, graph)
    return q*Dsqrt
```